Knowledge Intensive Planning

Marc Luria

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE
JUL 1988

2. REPORT TYPE

3. DATES COVERED
00-00-1988 to 00-00-1988

4. TITLE AND SUBTITLE
Knowledge Intensive Planning

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California at Berkeley,Department of Electrical Engineering and Computer Sciences,Berkeley,CA,94720

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This thesis describes a program called KIP (Knowledge Intensive Planner). KIP is a general, commonsense planner that can reason about planning situations in the real world for which it is provided information. KIP is the planning component of the UC (UNIX consultant) system. KIP is used to solve the problems the user poses to the UC. KIP has knowledge about UNIX commands, including the effects of those commands and under what conditions those commands can and should be issued. The best plan is reported to the user after (1) determining the goals of the user, (2) selecting and specifying a plan that fulfills the goals of the user (plan determination), (3) testing if the plan will work in this particular problem situation without causing unacceptable consequences, and (4) modifying this plan if necessary. A major problem in commonsense planning is the focus of attention on relevant knowledge. In particular, the problem of identifying potential plan failures in a plan is difficult, since there are often many sources of plan failure, both for failures due to an unsatisfied condition of a plan and failures due to goal conflict. This problem is further complicated because many values of conditions in a particular planning problem may be unknown. In order to address the problem of identifying potential plan failures, a new idea, called a concern, has been introduced. Concerns identify which aspects of a plan are most likely to fail.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**184**	

Knowledge Intensive Planning

Marc Luria

Abstract

This thesis describes a program called KIP (Knowledge Intensive Planner). KIP is a general, commonsense planner that can reason about planning situations in the real world for which it is provided information. KIP is the planning component of the UC (UNIX consultant) system. KIP is used to solve the problems the user poses to the UC. KIP has knowledge about UNIX commands, including the effects of those commands and under what conditions those commands can and should be issued. The best plan is reported to the user after (1) determining the goals of the user, (2) selecting and specifying a plan that fulfills the goals of the user (plan determination), (3) testing if the plan will work in this particular problem situation without causing unacceptable consequences, and (4) modifying this plan if necessary.

A major problem in commonsense planning is the focus of attention on relevant knowledge. In particular, the problem of identifying potential plan failures in a plan is difficult, since there are often many sources of plan failure, both for failures due to an unsatisfied condition of a plan and failures due to goal conflict. This problem is further complicated because many values of conditions in a particular planning problem may be unknown. In order to address the problem of identifying potential plan failures, a new idea, called a *concern*, has been introduced. Concerns identify which aspects of a plan are most likely to fail.

To Gayle

Acknowledgements

I would like to thank my advisor Robert Wilensky for his advice and support. My many discussions with him led to many of the ideas in this thesis. I wish to thank the other members of my thesis committee, Lotfi Zadeh - for being most gracious professor I have ever met, and Steve Palmer for providing a cognitive perspective on my work. I also wish to thank Richard Karp and Richard Fateman, the other members of my qualifying committee, for their guidance, and interest in my work and future plans.

The Berkeley AI Research Group has been a very exciting place to do research in the last few years. The following people contributed greatly to my ideas about planning and knowledge representation: (with M's first) Jim Martin, Jim Mayfield, Paul Jacobs, Lisa Rau, Margaret Butler, David Chin, Nigel Ward, Dan Jurafsky, Dekai Wu, Eric Karlson, Terry Regier, Rick Alterman, Charley Cox, Yigal Arens, Michael Braverman, and Sharon Tague. I would like to give special thanks to Peter Norvig for our many long discussion about KODIAK and everything else.

I would like to thank the sponsors of this research, which include the Defense Advanced Research Projects Agency (DoD), under Arpa Order No. 4871, monitored by Space and Naval Warfare Systems Command under Contract N00039-84-C-0089; the Office of Naval Research, under Grant No. 85-14890; the National Science Foundation under Grant No. 1ST-8514890; and the Alfred P. Sloan Foundation for a Cognitive Science Research Assistantship under Grant No. 83-12-1.

I would like to thank my parents Myron and Esther Luria for everything. My kids, Kobi and Nili, provided wonderful (and necessary) distraction. Most of all, I would like to thank my wife and best friend, Gayle, for taking responsibility for our lives during the last year and editing many versions of this thesis. Lastly, I'd like to thank Jim Martin for writing these acknowledgements.

Contents

List of Figures

Chapter 1

Introduction

1.1 KIP: Knowledge Intensive Planner

This thesis describes a commonsense planner called KIP (Knowledge Intensive Planner). KIP refers to two ideas: (1) KIP - the Theory, and (2) KIP - the Program. KIP - the Theory refers to the theory of planning presented in this thesis. The theory's primary motivation is the need to deal with the abundance of knowledge belonging to a commonsense planner. KIP's general strategy is to focus on those parts of the planning problem which are most important in a particular planning situation. KIP - the Program is an AI program designed to construct plans for UNIX operating system users as part of the UNIX consultant system. User goals are passed to KIP, and KIP returns a plan that the user can actually type to UNIX. The UNIX Consultant system is an intelligent natural language interface that allows naive users to ask questions about the UNIX operating system.

1.1.1 Planning in the UNIX Domain

Planning in the UNIX domain is difficult due to the large amount of detailed knowledge of a UNIX expert. In particular, a UNIX expert has much knowledge regarding the ways in which plans may fail. However, such experts only consider knowledge that is relevant to a particular planning situation. For example, suppose the user asks the following questions:

(1) `How do I print out a file named george?`

(2) `How do I print out John's file named george?`

In example, (2) a UNIX expert is likely to consider the read permission of the file named george. Since the user has specified that the file's owner is John, it is likely that

the user cannot read the file. However, in (1), the expert is not likely to consider the read permission of the file.

A UNIX expert knows about additional facts in the world that relate to use of UNIX commands. For example, suppose a professor asks the following question:

(3) `How do I print out my class midterm?`

In this example, a UNIX expert might realize that printing on the public printers would cause a conflict with the professor's goal of keeping the midterm secret. Therefore, it would suggest printing the midterm on the office printer which is reserved for such uses. However, the UNIX expert might also know that printing on the office printer is likely to be problematic in two ways: (1) the paper tray on the printer is often empty and (2) the office printer room is always locked. The UNIX expert is likely to consider these plan failures, but is unlikely to consider other plan failures such as (3) the printer must be turned on and (4) the computer network must be operating. The second set of plan failures are less likely to occur than the first set. However, the second set of plan failures might be important to consider in other planning situations.

1.1.2 UNIX Consultant

In this section, I present a short overview of the UNIX consultant system in order to better understand KIP's function in this system. The UNIX consultant (hereafter, UC) as a whole is best described in [38]. UC has five components which are currently invoked serially:

(1) Natural Language Parser (ALANA) -	Parses the user's natural language utterance into KODIAK knowledge representation form
(2) Goal Analyzer (PAGAN) -	Determines the user's actual goals by examining the user's utterance
(3) Agent (UCEgo) -	Determines UC's own goals
(4) Planner (KIP) -	Determines a plan for the user's goals
(5) Generator (MYGEN) -	Generates an answer to the user in English

For example, suppose the user asks the following question:

(4) `How do I delete the file named junk?`

ALANA [8] parses this English language sentence into a conceptual form. The PAGAN goal analyzer [25] determines that the user wants a plan for his goal of deleting the file named junk. UCEgo [6] determines that UC should address the user goal by constructing a plan for the user goal, and telling the user about the plan. KIP is given the goal of deleting the file named junk, and constructs a plan for the user goal, i.e. rm junk. The plan is then generated by the MYGEN natural language generator [38]:

(4) `To delete the file named junk, use rm junk.`

In addition, UC has many capabilities other than determining direct answers to user queries. For example, consider the following interaction from [6]. Suppose the user asks the following question:

(5) `What does ls -v do?`

UC responds:

(5) `There is no -v option for ls.`

UC has decided to ignore the user's direct question and shift its attention to the user's misconception. (For more information on such capabilities see [6].)

UC uses the KODIAK knowledge representation language [36] to represent UNIX concepts. (KODIAK is described in detail in Chapter 3.) The current implementation of KIP uses the most recent version of KODIAK. As of this writing, ALANA parser and UCEgo have yet to be ported to the new KODIAK. Therefore, simple versions of these programs have been written to demonstrate KIP's abilities. A parser, written by Peter Norvig, has been integrated into this version of the UC system. In addition, a simple intelligent agent program which always assumes that UC should construct a plan for the user's expressed goal has been implemented.

1.1.3 KIP's Role in UC

KIP determines plans for the user's goals which KIP receives as input. Actually, KIP does not differentiate between its own goals and the goals of the user. Thus, the fact that KIP plans for the user's goals does not make KIP's planning task different from that of the previously discussed planners.

KIP's process of constructing a plan for the user's goals is called *plan synthesis*. Plan synthesis is an iterative process composed of three parts:

(1) *goal establishment* - establishment of those goals KIP needs to address

(2) *plan determination* - determination of a plan for those goals

(3) *plan failure detection* - detection of potential plan failures

During goal establishment, KIP decides which of the user's goals it will try to satisfy. Many of these goals come from the PAGAN goal analyzer[25]. During plan determination, KIP selects a potential plan for the selected goal and specifies the plan for the particular planning situation. KIP tests the plan to determine whether it will really satisfy the user goal in this particular planning situation during the plan failure detection phase. The two ways that a plan may fail are termed condition failure and goal conflict failure. If a certain condition needs to be satisfied in order to execute the plan, the satisfaction of the condition becomes a new goal that needs to be satisfied. If the determined plan causes a conflict with another user goal, the resolution of the goal conflict becomes a new goal. Finally, KIP needs to determine if all the user goals have been satisfied. If all user goals are not satisfied, KIP iterates through the process again. In this way, a plan is gradually synthesized. KIP's plan synthesis algorithm is described in detail in Chapter 4. Chapter 5 focuses on goal establishment and Chapter 6 focuses on the plan determination process. Plan failure detection is discussed in Chapters 7-10.

1.2 Problem-Solving in Knowledge Intensive Domains

Early work in artificial intelligence planning developed in what I will call *knowledge-deficient* domains (cf. weak method problem solving, [29]). Given a particular problem state, the knowledge-deficient planner has very little specific knowledge of how to proceed. A knowledge-deficient planner, (e.g. one that uses means-end analysis [12,29,31]) might merely select the plan that results in a state closest to the goal state using only knowledge about a few operators.

In contrast, most human problem solving seems knowledge-intensive. People acquire much knowledge about the many plans they encounter in everyday life. I call a planner whose operation is based on knowledge a *commonsense planner*[35]. A commonsense planner should be able to deal with a large body of commonsense knowledge about a particular domain. Such knowledge includes a general understanding of planning strategy, detailed descriptions of plans, descriptions of the conditions necessary for these plans to execute successfully, and descriptions of potential goal conflicts.

For example, suppose that a robot planner is faced with the classic AI problem of the monkey and the bananas proposed by McCarthy[26]. In this problem, a monkey is placed in a room containing only a box and bananas hanging from the ceiling. In order to reach the bananas, the monkey must stand on the box.

Knowledge-deficient planners such as GPS[10] represent the planning knowledge as the simple operators that can be executed. For example, the monkey can change

the location of the box by moving it and the monkey can change his own location by walking. A knowledge-deficient robot planner constructs a plan out of these simple operators. In contrast, most humans know that standing on something near an object is an effective means of retrieving an out of reach object. Thus, it is not surprising that the human plans how to reach bananas more easily than does the robot planner deprived of such experience.

Commonsense planners have just the opposite problem of knowledge-deficient planners: there is an abundance of knowledge about the domain. The central issue in commonsense planning is the determination of those parts of the knowledge-base relevant to the planner's current problem solving task. Only once the planner focuses its attention on those relevant pieces of knowledge can it use this knowledge to plan effectively.

1.3 Plan Failure Detection

One particularly important issue is the need to focus the planner's attention on those parts of a potential plan most likely to cause plan failure. Since a commonsense planner has much knowledge about how a particular plan might fail, determining such knowledge relevance is a difficult task.

For example, let us return to the monkey and bananas problem discussed above. GPS's description of this problem modeled the following conditions:

- in order to grasp the bananas, the monkey must be on top of the box and under the bananas
- in order to climb on top of the box, the monkey must be next to the box
- in order to move the box, the monkey must be next to the box

These are important conditions which the planner must know in order to construct a plan that will allow him to get the bananas. Indeed, the problem was constructed in such a way so that few conditions need to be satisfied in order to execute the plan successfully. However, a commonsense planner might also know about the following conditions:

- in order to move the box, the monkey must be on the floor (i.e. not on the box)
- the height of monkey and the height of the box must be more than the height of the bananas (or the monkey will not be able to reach the bananas)
- the box must be made of sturdy materials (or the monkey will fall through the box)
- the box must not be too heavy for the monkey to move
- the monkey's hand must be empty (so the monkey can hold the bananas)

The second set of conditions are less likely to cause plan failure than the first set of conditions. However, it is still important for a commonsense planner to be aware of the second set of conditions in certain planning situations when the conditions do fail. In fact, the planner may know about many more conditions which are even less likely to cause plan failure. For example,

- there must be oxygen in the room
- the monkey must be free to move about the room

Furthermore, there are many plans for getting the bananas might be problematic:

- the monkey might hurt himself if he falls off the box
- the monkey might get sick after eating all the bananas
- a large bunch of bananas might fall on the monkey and kill him

Thus, even for this relatively simple planning scenario, a commonsense planner would know about a large number of potential plan failures. Plan failure detection is a major focus of this thesis.

1.4 Properties of a Commonsense Planner

In this section I discuss five properties of a commonsense planner. There are many different ways a commonsense planner (hereafter, CSP) might address the planning problem. These properties are particularly useful in demonstrating the problems a CSP algorithm must address in plan failure detection. They are also used as criteria for assessing a potential algorithm for CSP.

The five properties are:

(1) *Knowledge Rich* -	use of extensive knowledge regarding plans and goals
(2) *Default Situation Knowledge* -	reliance on default situation knowledge when planning scenario is not completely defined
(3) *Cognitive Validity* -	attempt to model human behavior
(4) *General Knowledge Application* -	use of high-level knowledge in specific situations

(5) *Knowledge Efficient* - access of only relevant knowledge

The most important consideration for any potential algorithm of a commonsense planner is the knowledge base that is necessary to use this algorithm. Therefore, in the context of the following discussion of these five properties, I also make parallel assumptions about the properties of a CSP's knowledge base.

1.4.1 The Knowledge Rich Property

According to the knowledge rich property, CSP's knowledge base should have detailed knowledge regarding plans and goals. For example, a CSP should know which plans are appropriate for which goals, all the conditions necessary for plans to execute successfully, and many potential goals. Therefore, a CSP algorithm must have the ability to manipulate this knowledge effectively. For example, suppose that a CSP has knowledge that a certain plan is the best plan for a particular goal. If a CSP is asked to satisfy that goal, then that best plan should be selected. The algorithm and knowledge base should interact such that regardless of the size of CSP's knowledge base, the relevant knowledge is always considered.

1.4.2 The Default Situation Knowledge Property

In many planning situations, all the values of the parameters of a plan may not be known. When the actual values of these parameters are not provided in the description of a particular planning problem, a CSP needs to rely on default knowledge. The default values for these parameters may change according to the context of the particular planning situation. Therefore, a CSP needs a mechanism to manipulate the default knowledge so as to determine the default values for these parameters in unique problem situations.

1.4.3 The Cognitive Validity Property

When discussing an algorithm for a CSP, I say that an algorithm seems cognitively valid, or conversely, that an algorithm does not seem to address a problem the way that people do. The *cognitive validity* property refers to the degree to which a CSP algorithm corresponds to a human planner in its problem solving strategies. Since our understanding of human planning is so limited, cognitive validity is the most difficult property of a commonsense planning algorithm to evaluate.

1.4.4 The General Knowledge Application Property

A CSP should be able to create plans for goals in particular planning situations. Unless a CSP has a plan for the exact situation described in the user's plan, a CSP should

apply general knowledge in a specific planning situation. For example, a CSP might have planning knowledge about grasping bananas, but it probably would not have knowledge about grasping bananas in Evans Hall. It is impossible to store plans for every potential planning situation. Therefore, any algorithm that addresses a CSP should have the ability to represent general knowledge about plans and goals and to use that knowledge in particular situations. However, when specific knowledge about a particular situation is important, this specific information should be included in the planner knowledge base. According to Wilensky's First Law of Knowledge Application [35], the planner should apply the most specific piece of knowledge available.

1.4.5 The Knowledge Efficient Property

According to the knowledge efficient property, a CSP should access only pieces of knowledge which are relevant to the particular planning situation. It is useful to discuss the knowledge-efficient property in terms of the previous four properties. A CSP algorithm that considers all the knowledge of a knowledge-rich planner might cause a combinatorial explosion. A Knowledge Efficient planner mimics a human planner who generally considers only those pieces of knowledge which are relevant to the particular problem under consideration. A CSP should only consider default situation knowledge for unspecified parameter values which are relevant to the current planning situation. General knowledge should only be applied to a particular planning situation if that knowledge is relevant.

The knowledge-efficient property is the one referred to most in an evaluation of potential algorithms for a CSP. There are two reasons for this predominance. Firstly, the knowledge-efficient property is related to the other properties discussed. An efficiency component is inherent in each of the other properties. More importantly, knowledge efficiency is the central problem in building an algorithm that will work on a computer.

1.5 Concerns

The focus of this thesis is on plan failure detection. As discussed in the previous section, a commonsense planner needs a means of limiting those plan failures under consideration. The plan failure detection problem is discussed in detail in Chapter 7. The goal conflict detection problem, discussed in Chapter 9, is even more complex, due to the large numbers of goals that a user might have. In Chapter 8, a new concept, called a *concern*, is introduced in KIP in order to address this problem.

A concern refers to those aspects of a plan which should be considered because they are possible sources of plan failure. A concern describes which aspects of a plan are likely to cause failure.

Stored concerns are stored in KIP's long term knowledge-base. These include important conditions of a stored plan that need to be considered when suggesting a possible plan, and descriptions of potential goal conflicts with the effects of a stored plan. Thus, stored concerns are a way for the planner database designer to express his personal experience regarding which aspects of a stored plan are most likely to fail.

Dynamic concerns arise during the planning process itself. They are usually instances of stored concerns. When a possible plan is considered that is an instance of some previously known stored plan, the stored concerns of that stored plan generate dynamic concerns for the new instance of the plan. When KIP notices a potential condition or goal conflict failure, but has not yet decided whether such a failure will occur, KIP instantiates a dynamic concern.

There are a number of different ways in which concerns can be characterized. Chapter 8 focuses on *condition concerns*, i.e. concerns about conditions of a potential plan. *Goal conflict concerns*, discussed in Chapter 10, refer to concerns regarding conflict between the effects of a potential plan and a user goal.

In the rest of this section, I present a KIP example where both condition concerns and goal conflict concerns are detected. Trace output is printed in normal font, my annotations to the trace are in italics.

Figure 1.1: KIP Trace of List Directory Contents on a Sun

How do I find out the files in the directory named /usr/local on my sun?

;; KIP receives the following goal as input from the parser and the PAGAN goal
;; analyzer:

```
(list-directory-effect-1
 (Information-About-Of-List-Directory-Effect-30
  (dir-2
   (File-Name-26 /usr/local-1)
   (Owner-28 root-user-27)
   (Directory-Contents-107 directory-list-69)))
```

;; KIP has been asked to provide the user with information about the directory
;; named /usr/local. The parser determines that this directory is a type of
;; system file, since its name begins with /usr. Therefore, this directory's owner
;; is root, i.e. the UNIX system administrator. The contents of this directory is
;; directory-list-69, which is a list of files.

```
 (Experiencer-Of-Unix-Information-Effect-64 uc-user-1)
```

;; The Experiencer of this effect is the UC user.

```
 (Initial-State-Of-List-Directory-Effect-62
  (knows-directory-contents-state-40
   (Value-Of-Knows-Directory-Contents-122 directory-contents-state-124)
   (Object-Of-Knows-Directory-Contents-51 uc-user-1)))
 (Final-State-Of-List-Directory-Effect-87
```

(knows-directory-contents-state-65
(Value-Of-Knows-Directory-Contents-106 directory-contents-state-67)
(Object-Of-Knows-Directory-Contents-76 uc-user-1))))

;; *Information effect, like other state changes that KIP knows about, are*
;; *represented as state changes. In this case, the final-state is that the user knows*
;; *the contents of the directory to be* directory-contents-state-67, *while in the*
;; *initial-state, the user knows the contents of the directory to be*
;; directory-contents-state-124. *The representation of state-changes and other*
;; *KODIAK objects is described in detail in Chapter 3.*

Entering Goal Establishment Phase:

KIP is trying to determine a plan for the list of goals:

(list-directory-effect-1)

Selecting a goal from the List Of Goals ((list-directory-effect-1))
selecting the remaining goal
list-directory-effect-1

;; *KIP first has to select a particular goal to address. In this case, KIP only has*
;; *one goal from which to select. In more complex situations, where KIP has more*
;; *than one goal, KIP must select one of these goals. Goal selection is described in*
;; *Chapter 5.*

Entering Plan Determination Phase:
Looking for a plan for the Current Goal (list-directory-effect-1)

First looking at stored plans

Selected ls-command as a potential plan

;; *KIP has examined its knowledge base of stored plans for plans that satisfy the*
;; *goal of list-directory-effect. It has selected the ls-command for this goal.*
;; *Plan selection is described in detail in Chapter 6.*

Now specifying the plan for the particular planning situation:

;; *KIP must now specify the plan for this particular planning situation. Plan*
;; *Specification refers to the specification the values of the general plan for the*
;; *particular problem situation. Plan Specification is described in*
;; *Section 6.2.1.*

(ls-command-on-sun-161

(Machine-Of-Ls-Command-On-Sun-159 sun-23)

;; *Since KIP is told in its input that the list directory effect must be on a*
;; *sun, KIP specifies that the machine on which this ls-command command is*
;; *executed must be a sun. This causes a concretion inference. Concretion*
;; *inferences[30] refer to inferences that an individual is dominated*
;; *by one of the subtypes of its parent. In this case, KIP concretes its*
;; *ls-command plan to the more specific ls-command-on-sun plan. This*
;; *concretion is useful because specific facts are known about*
;; *ls-command-on-sun that are not known about ls-command. Specifically,*
;; *KIP knows about particular types of plan failures when the ls command is*
;; *executed on a sun. Concretion inferences are described in Section 5.2.4.*

(Ls-Directory-134 dir-2)

;; KIP specifies that this ls-command directory argument is dir-2. This
;; specification is made due to knowledge equating the directory argument of the
;; ls-command with the experiencer of the list-directory-effect. This is done using
;; an equate relationships. Equate relationships are KODIAK which specify that the
;; the ranges of two relations are always equal. Equates are described in detail in
;; Section 3.5.

(Owner-Of-File-Arg-136 root-user-27)

;; the owner of the directory is a root user

(Format-Of-Unix-File-Command-140
 (unix-file-command-format-139
  (Command-Arg-144 ls-string-141)
  (Format-File-Arg-146 /usr/local-1)))

;; The format specifies the way in which the user could actually execute the
;; plan by typing: ls /usr/local

(Intended-Effect-Of-Ls-Command-132 list-directory-effect-1)

;; This is the same list-directory-effect-1 *that was passed to KIP as input*

(Actor-Of-Unix-Information-Command-150 uc-user-1))

;; This is the same uc-user who wanted to know the contents of the
;; /usr/local directory

Entering Plan Failure Detection Phase:

The Current Value (root-user-27) of Owner-Of-File-Arg-136 cannot be the Desired Value (uc-user),

so a violated default is detected

;; KIP has detected a default violation. Normally the owner of the directory is
;; just a uc-user. However, in this case, the directory is owned by the root
;; user. Due to this violated default, KIP considers violated default concerns.
;; Violated Default concerns are describe in detail in Section 8.5.

The Violated Default Concern (not-owner-violated-default-concern) is detected

;; This is a general violated default concern which describes the class of
;; concerns where the user is not the owner of a file or a directory. There are
;; two violated default concerns which are instantiatiated in this particular
;; planning situation:

Therefore the Individual Violated Default Concern
(ls-sun-must-be-in-directory-concern-220) is instantiated

;; This is a violated default condition concern regarding a condition of the
;; potential plan. The concern reflects a potential plan failure that occurs
;; when a user executes an ls command on a directory which is not in the
;; same file system as the user's current directory. (The failed ls command
;; never returns.) It is unlikely that files owned by the user are in separate
;; file systems. Even expert users are not usually aware of the
;; particular directory's file system. Therefore, this concern is classified in the
;; set of concerns which are instantiated when the file argument is not owned
;; by the user. This plan failure only occurs when the ls-command is
;; executed on a sun computer. Therefore, this specific concern is related to

;; the ls-command-on-sun and not to ls-command.
;; Specific concerns are described in Section 8.3.2.

Therefore the Individual Violated Default Concern
(too-much-output-concern-167) is instantiated

;; This a goal conflict concern between the effect of the specific ls-command
;; and a goal of the user. The ls command might cause a large amount of
;; output to appear on the user's terminal. This effect conflicts with the user's
;; goal of seeing the output. Actually, goal conflict concerns refer to conflicts
;; between plan effects and user interests. Interests are general
;; states that KIP assumes are important to the user. Interests are described in
;; Section 5.2.3.

;; The ls-command command inherits this concern from the
;; unix-command-with-output category. This category refers to the class of
;; unix-commands which produce output.

;; KIP next evaluates these two concerns in this particular planning situation:

Evaluating the condition concern: ls-sun-must-be-in-directory-concern-220
The condition of concern is the current-directory-state-226 of the
uc-user-1
The current value of the condition is directory-228
The desired value is dir-2

;; This concern is only applicable if KIP knows that the user's current
;; directory is not already dir-2.

Current Value (directory-228) is less specific than the Desired Value (dir-2)

;; directory is the superordinate category of dir-2, *and* directory-228 *is*
;; generated as a placeholder for directory.

Try to make the current value more specific using defaults
Default-value is anything.

;; KIP has no default knowledge about the user's current directory. Therefore, KIP
;; decides to use the information it already has, i.e. the current directory is an
;; individual directory. No other information about this directory is known.

Since the Current Value (directory-228) is more specific than the Default Value (anything),
Instantiate concern that current value be changed

Creating a goal that reflects a change from the Current Value (directory-228)
to the Desired Value (dir-2)

Creating the goal:
(change-current-directory-291
 (New-Directory-289
  (dir-2
   (Owner-28 root-user-27)
   (File-Name-148 /usr/local-1)))

;; The new-directory is dir-2 which is the /usr/local directory

(Initial-Value-Of-Normal-State-Change-253 directory-228)

;; The old directory is directory-228

(User-Of-301 uc-user-1)

(Final-State-Of-Change-Current-Directory-303
 (current-directory-state-285
  (Object-Of-Current-Directory-308 uc-user-1)
  (Value-Of-Current-Directory-272 dir-2)))
(Initial-State-Of-Change-Current-Directory-292
 (current-directory-state-226
  (Object-Of-Current-Directory-297 uc-user-1)
  (Value-Of-Current-Directory-279 dir-288))))

;; The final state and initial state are created using equate relationships
;; defined for this state-change. In the initial-state, the user's
;; current-directory is dir-288. *In the final-state, the user's*
;; current-directory is dir-2.

Asserting the fact that the final-state of the goal (current-directory-state-285)
starts before the start of plan interval (ls-command-on-sun-161)
So that the condition holds before the plan is executed

;; KIP knows that the user must change his current directory before the
;; ls-command is executed.

;; Before KIP determines a plan for the change-current-directory-291 *goal, it*
;; first evaluates the other concern KIP is considering.

Evaluating the goal conflict concern: too-much-output-concern-167

This Concern (too-much-output-concern-167) potentially conflicts
with the users Concern Interest (output-scroll-off-screen-186)

This interest is applicable depending on the Concern Condition
(directory-size-state-171) of dir-2

;; KIP is evaluating the output-scroll-off-screen-186 *interest in this particular planning*
;; situation. The interest evaluation depends on the size of the
;; directory represented as a directory-size relation.

The current value of the condition is size-constant-173
The undesirable value is large
Current Value (size-constant-173) is less specific than the Undesirable Value (large)

;; KIP has no specific knowledge about this particular directory.

Try to make the current value more specific using defaults
Default-value is large.
The Current Value (size-constant-173) is less specific than the Default Value (anything),
Therefore, the default value will be used.

;; KIP has consulted its context-dependent default mechanism and determined
;; that the default value of the size of this directory is large. This default
;; knowledge is based on the fact that directories owned by root
;; tend to be large, while individual users generally have small directories.

Kip is now determining whether a goal should be instantiated to reflect the user
Interest (output-scroll-off-screen-186)
Since the degree of concern is high a goal is instantiated which reflects the threat to this interest
In this planning situation

;; KIP has instantiated a goal, based on the threat to the user's interest.

;; The goal is to prevent the scrolling of the output of the plan which KIP is
;; constructing.

Creating the goal:
(no-scroll-off-screen-245
(Experiencer-Of-No-Scroll-Off-Screen-197 output-stream-190)
;; output-stream-190 *is the output-stream created by* ls-command-on-sun-161
(Final-Value-Of-No-Scroll-Off-Screen-208 true-209)
(Initial-Value-Of-No-Scroll-Off-Screen-218 false-219)
(Final-State-Of-No-Scroll-Off-Screen-204
(on-screen-state-198
(Value-203 true-209)
(Object-202 output-stream-190)))
(Initial-State-Of-No-Scroll-Off-Screen-216
(on-screen-state-210
(Value-215 false-219)
(Object-214 output-stream-190))))

;; The initial-state is that the output goes off the screen, while the final-state
;; is that the output does not scroll off the screen.

Asserting the fact that the final-state of the goal (on-screen-state-198)
is after the plan interval (ls-command-on-sun-161)
So that the goal conflict is avoided after the command is executed.

Entering Goal Establishment Phase:

;; This is the beginning of KIP's second iteration of plan synthesis.
;; In this iteration, KIP is addressing goals that were generated
;; to reflect concerns that KIP has about the ls-command-on-sun-161 *plan.*

Selecting a goal from the List Of Goals ((no-scroll-off-screen-245
change-current-directory-291))

;; KIP is now selecting a goal from among those goals generated by concerns
;; about the ls-command-on-sun-161 *plan.*

Sorting the goals according to importance level
The List Of Goals ((no-scroll-off-screen-245 change-current-directory-291))
is already sorted in order of importance

Entering Plan Determination Phase:
Looking for a plan for the Current Goal (no-scroll-off-screen-245)

First looking at stored plans

Selected more-filter-command as a potential plan

;; *the* more-filter-command *is a plan for preventing output to*
;; scroll off the screen.

Now specifying the plan for the particular planning situation:
(more-filter-command-329
(Filter-Input-332 output-stream-190)
;; output-stream-190 *is the output of the* ls-command-on-sun-161 *plan. This is the*
;; output that KIP wants to prevent from scrolling off the screen.
(Machine-Of-Of-Unix-Command-339 sun-23)

(Intended-Effect-Of-More-Filter-Command-330 no-scroll-off-screen-245))

;; *This command is a filter. The output of the* ls-command-on-sun-161 *plan is piped*
;; *to the more command and output will not scroll off the screen.*

Asserting that more-filter-command-329 comes after ls-command-on-sun-161

;; *Since KIP knows the* no-scroll-off-screen-245 *goal must be satisfied after*
;; ls-command-on-sun-161, *it knows that the plan constructed to satisfy*
;; no-scroll-off-screen-245 *must be executed after* ls-command-on-sun-161.

Entering Plan Failure Detection Phase:
No plan failures detected for Candidate Plan (more-filter-command-329)

Entering Goal Establishment Phase:
Selecting a goal from the List Of Goals ((change-current-directory-291))
selecting the remaining goal
change-current-directory-291

;; *This goal was generated in response to a condition concern regarding the*
;; *user's current directory.*

Entering Plan Determination Phase:

Looking for a plan for the Current Goal (change-current-directory-291)

;; *KIP now specifies this goal for the present planning situation. The*
;; *representation of* change-current-directory-291 *is shown on page 12.*

First looking at stored plans

Selected cd-command as a potential plan
Now specifying the plan for the particular planning situation:

```
(cd-command-347
 (Intended-Effect-Of-Cd-Command-348 change-current-directory-291)
 (Cd-Directory-350 dir-2)
```

;; dir-2 *is the /usr/local directory*

```
 (Actor-Of-Unix-Command-369
  (uc-user-1
   (Current-Directory-296 directory-228)
   (Current-Directory-280 dir-2)))
```

;; *The user has two current-directory relations.*

;; Current-Directory-296 *is the user's old current-directory, while*
;; Current-Directory-280 *is the new directory.*

```
 (Format-Of-Unix-File-Command-356
  (unix-file-command-format-355
   (Command-Arg-360 cd-string-357)
   (Format-File-Arg-362 /usr/local-1))))
```

;; *The user could execute this command by typing cd /usr/local*

Asserting that cd-command-347 comes before ls-command-on-sun-161

;; *This is based on knowledge that the* change-current-directory-291 *goal needs*
;; *to start before the* ls-command-on-sun-161 *plan.*

Entering Plan Failure Detection Phase:
No plan failures detected for Candidate Plan (cd-command-347)

;; The plan can now be suggested to the user:

To find out what files are in the directory named /usr/local, type ls /usr/local.
However, you should first change the current directory to the directory named /usr/local, by typing cd /usr/local.
Since the output is likely to scroll off the screen, pipe the output of the ls command through the more filter.

;; KIP first generates the main part of the plan, i.e. using the ls command. Then,
;; KIP generates those subplans that addresses concerns about the plan that must be
;; addressed before the ls command is executed. Finally, KIP generates concerns
;; that must be addressed after the ls command is executed.

1.6 Conclusion

In this chapter, I have characterized the planning problem for a commonsense planner. The central issue in commonsense planning is the determination of those parts of the knowledge-base relevant to the planner's current problem solving task. One particularly important task is the determination of potential plan failures in a selected plan. Since a commonsense planner knows about many ways a potential plan may fail, the determination of concept relevancy is particularly important for plan failure detection. I have also described a number of properties of a commonsense planner.

In introducing KIP, I have focused on KIP's means of detecting potential plan failures. Concerns, a methodology for dealing with potential failures of plans, has been described. Concerns focus the attention of KIP so that it only considers those conditions and goals which are likely to cause plan failure. In Chapters 7-10, plan failure detection and concerns are discussed in greater detail.

Chapter 2

Related Research in Planning

Planning research is one of the oldest research topics in Artificial Intelligence. In this section, I discuss two types of planners:

(1) Classic AI Planners - Planners which view planning as search

(2) Goal Based Planners - Planners which evolved from an understanding of goals

2.1 Classic AI Planners

2.1.1 GPS

One of the earliest planners was GPS, or General Problem Solver [10,29]. GPS introduced a planning strategy termed *means-end analysis*. Means-end analysis entails selecting the plan that reduces the greatest difference between the present state and the goal state. GPS first searches for differences between the goal state and the present state. It then utilizes a *difference table* that indexed the known operators by the difference they reduced. GPS selects the operator that reduces the greatest amount of difference. Once a selected operator is applied, a new state is created. GPS then recursively applies its planning algorithm to this new state.

Greatest difference is determined by using a precomputed difference table. Differences are reduced in order of difficulty according to a predetermined ordering, termed a DIFF-ORDERING. The DIFF-ORDERING is assigned by the GPS implementor based on the ease of task accomplishment.

For example, let us consider an example from [10] wherein GPS determines a plan for the monkey and bananas problem discussed in Section 1.3. The difference between the goal state and the initial state is that the monkey has the bananas in its hand. GPS reduces

this difference by applying the GET-BANANAS plan to the initial-state. GPS must then reduce two more differences, (1) the monkey must be on the box and (2) the box must be under the bananas. GPS has no plan which will reduce both differences. Since difference (2) is stored as being more important than difference (1) in the DIFF-ORDERING, difference (2) is reduced first by using the MOVE-BOX plan. However, in order to execute the MOVE-BOX plan, the monkey must be next to the box by using WALK. After the monkey moves the box, difference (1) - monkey on the box, must still be reduced. Therefore, the monkey CLIMBs on the box. At this point, the monkey can get the bananas and a complete plan is returned (1) WALK to box, (2) MOVE-BOX under bananas, (3) CLIMB box, and (4) GET-BANANAS.

2.1.2 STRIPS

Fikes and Nilsson [12] used means-end analysis in a robot planner called STRIPS to create plans for a mobile robot. STRIPS substitutes logic for the operator difference table used in GPS. Therefore, STRIPS avoids the necessity of computing a difference table. STRIPS is presented with a well-formed formula describing the goal state and the present state as well as a set of formal descriptions of available operations. STRIPS then attempts to prove the truth of the goal state using a resolution theorem prover. If an individual subgoal of the goal state cannot be proven from the present state, STRIPS selects an operator that will allow the proof attempt to continue by reducing the greatest number of clauses. The preconditions of the selected plan then become the new clauses that STRIPS must prove.

Unlike GPS, STRIPS lacks a method for the determination of relative plan importance. For example, suppose that STRIPS is faced with the monkey and bananas task. STRIPS might choose to first reduce difference (1) (monkey on box). Thus, STRIPS would plan to WALK to the box, and CLIMB the box. Then STRIPS would plan to reduce difference (2) (box under bananas). However, using MOVE-BOX requires the monkey to be on the floor. Thus, STRIPS would plan for the monkey to DESCEND from the box, move the box under the bananas, climb the box again, and then get the bananas. STRIPS would then return the following suboptimal plan: (1) WALK to box, (2) CLIMB box, (3) DESCEND box, (4) MOVE-BOX under bananas, (5) CLIMB box, and (6) GET-BANANAS. This suboptimal plan would be created because the CLIMB box plan deletes a precondition of the MOVE-BOX plan. The interaction between the precondition of the MOVE-BOX plan and the effect of the CLIMB plan is termed an *interacting subgoal*.

Another problem with STRIPS is the size of the search space. STRIPS might search a combinatorially large search space in order to find a plan. As the size of the STRIPS knowledge base of plans and preconditions grows, the processing time grows exponentially.

2.1.3 ABSTRIPS

ABSTRIPS [31] modified STRIPS in order to avoid the problems of large search spaces and interacting subgoals. ABSTRIPS is termed a hierarchical planner because it plans in a hierarchy of abstraction spaces. In other words, it creates a plan at a particular level of abstraction. If that plan is successful, it then fills out the plan at the next lower level of abstraction. ABSTRIPS utilizes GPS's idea of reducing the *important* differences in the problem first, by assigning criticality levels to differences. ABSTRIPS determines a plan for those clauses at the highest level of criticality by first reducing those differences with the highest criticality. Criticality levels are assigned by the program itself. The assignment is based on the difficulty inherent in satisfying the preconditions for plans to reduce a particular difference.

For example, in the monkey and bananas problem, ABSTRIPS could assert that difference (2) - box under bananas is of higher criticality than difference (1) - (monkey on box). ABSTRIPS would thus avoid the problems that STRIPS encountered in this example.

ABSTRIPS's planning strategy is particularly important when one part of the potential plan fails. Suppose there is no way to satisfy the preconditions of a particular plan, ABSTRIPS often realizes this at a high level of abstraction and avoids the construction of a complete plan. For example, suppose that ABSTRIPS knew that the box was unmovable and not under the bananas. Since the mobility of the box has a high level of criticality for the GET-BANANAS plan, ABSTRIPS would select another potential plan for retrieving the bananas.

2.1.4 NOAH

NOAH [32] avoided problems of interacting subgoals by using a least commitment strategy. Unlike [AB]STRIPS, NOAH does not specify the order of occurrence for the steps in a plan until all plan steps are selected. Once NOAH has determined a complete plan for each of its goals, it then makes decisions regarding the ordering of the plan steps. NOAH plans by constructing and refining a procedural network. It initially assumes that two subgoals can be solved by different subplans in parallel, and then later commits to a particular ordering of plans. Plan ordering is accomplished through the use of *critics*. These critics examine the various plan steps and detect negative interactions. For example, the RESOLVE-CONFLICT critic notices when a postcondition of one subplan removes the precondition of another subplan.

NOAH is designed for problems like the monkey and bananas problem. If NOAH were faced with such a planning task, it would construct plans for both the box under bananas and monkey on box states. The RESOLVE-CONFLICT critic would then determine that the CLIMB box plan deletes a precondition of the MOVE-BOX plan, necessitating an ordering of plans.

In order to detect conflicts, NOAH computes a TOME, a table of multiple effects, each time a new action is added to the plan. This table includes all preconditions which are asserted or denied by more than one step in the current plan. Conflicts are recognized when a precondition for one step is denied in another step. In order to construct this table, NOAH must enter all the effects and preconditions for each of the steps in the plan every time a new step is added to the plan.

NOAH'S separation of Goal Conflict Detection Phase from the rest of the planning process was an important addition to planning research. However, NOAH'S approach is problematic. It only detects conflicts that occur as a result of deleted preconditions. Other conflicts, such as conflicts between effects of a plan and other goals, cannot be detected using this method. If many planner goals were included in a TOME, as would be necessary in real world planning situations, this method would be computationally inefficient.

2.1.5 Relationship of Classic AI Planners to KIP's Approach

The means-end analysis planners discussed in this section test all the conditions of a plan before returning a plan. This method is not problematic when the planners are applied to knowledge-deficient problems with a limited knowledge of conditions. The plans they consider have only a few conditions.

For example, if the STRIPS [12] robot uses a plan to open a door between two rooms, STRIPS checks whether the robot is next to an object, the object is a door, and the door is closed. But STRIPS does not check whether the door handle is working or the hinges are properly attached. The last two states should also be conditions of OPEN-DOOR plan. If they are unsatisfied, the plan will not work. However, STRIPS does not consider these states to be conditions. If STRIPS included these conditions as conditions of the OPEN-DOOR plan, the STRIPS combinatorial explosion would worsen greatly. If KIP had a similar plan, these conditions would be stored as conditions of the OPEN-DOOR plan. However, they would not usually be considered since they are not concerns of the OPEN-DOOR plan.

GPS [10], ABSTRIPS [31], and NOAH [32] considered the conditions of a plan in order of difficulty. They thus modified the plan accordingly in different abstraction spaces. For example, in the OPEN-DOOR plan, ABSTRIPS assigns the object-is-a-door condition a higher criticality level than the robot-is-next-to-the-door condition. The first condition cannot be changed by any plan, while the second condition can be changed by the GOTO plan. All preconditions of the highest criticality level are considered in the highest abstraction space, and those of a lower criticality level are considered in a lower abstraction space.

By planning in a hierarchy of abstraction spaces, ABSTRIPS reduced some of the combinatorial explosion inherent in STRIPS. However, if more conditions were added to their plan descriptions, the use of abstraction spaces would not address the problem of an

even greater number of possibilities. For example, if ABSTRIPS included the door-hinge and door-handle conditions as conditions of the OPEN-DOOR plan, both of these conditions would need to be considered before the robot-is-next-to-the-door condition. They both are more difficult to attain than the robot-is-next-to-the-door condition. As these additional conditions are difficult to achieve, ABSTRIPS would always need to consider them first, thus increasing the combinatorial explosion. The problem with both ABSTRIPS and GPS is that they use difficulty as a metric for importance. Thus, they might deal with the difficult parts of a plan before dealing with the parts of a plan that are likely to fail. According to KIP's approach to plan failure, only the most likely plan failures are considered.

2.2 Goal-Based Planning

A very different approach to planning resulted from work regarding the characterization of plans and goals for natural language understanding. Schank [33] suggested that understanding the plans of an agent was an important task for story understanding. Wilensky's [37] PAM system understood simple stories by detecting the plans of actors. PAM was not a planner, it was only meant to understand plans. Thus, PAM had knowledge about how people plan in problem situations. While the classic AI planners planned in situations that were easy to formalize, e.g. the blocks world, PAM understood plans in real-world situations which are much more difficult to formalize, e.g. threatening, slaying dragons, eating. The approach led to a number of different planners discussed in this section.

2.2.1 PANDORA

In [35], Wilensky describes the PANDORA planner, devised to expand on the ideas in PAM and apply them to planning. There are four main components of PANDORA:

(1) Goal Detector -	determines planner goals from planning situation
(2) Plan Proposer -	find stored plans which are relevant to detected goals
(3) Projector -	builds hypothetical world models based on execution of proposed plans
(4) Executor -	performs specified action in the world

Wilensky's work focuses on the identification of different goal interactions. He introduces four goal relationships:

(1) Goal Conflict -	Conflicting goals held by the same individual
(2) Goal Overlap -	Overlapping goals held by the same individual
(3) Goal Competition -	Conflicting goals held by different individuals
(4) Goal Concord -	Overlapping goals held by different individuals

Goal Conflict and Goal Competition are termed negative interactions, while Goal Overlap and Goal Concord are termed positive interactions. These goal interactions are important for both understanding stories about goal interactions and planning in situations where goals interact.

Wilensky also proposed a theory of metaplanning, i.e. planning using knowledge about the planning process itself. For example, metaplanning knowledge would dictate a general plan strategy when faced with one of the four goal relationships discussed above.

Wilensky's work has provided a basic framework and vocabulary for the study of commonsense planning. In KIP, I have focused on the first type of goal interaction, i.e. goal conflict.

2.2.2 CHEF

Hammond's [13] CHEF planner creates plans (recipes) for use in Szechwan cooking. CHEF is a case-based planner, i.e. planning based on memory of previous planning cases. CHEF selects a plan by searching for a previous case, and then revising that plan for the particular planning situation. An important focus of Hammond's work is in on the selection of plans based on avoided plan failures. For example, in [13] CHEF realizes that a BEEF-AND-BROCOLLI recipe fails because the brocolli is soggy. Since CHEF is aware of this type of failure, it uses a stored plan that avoids the soggy vegetable problem. CHEF also includes a learning component so as to anticipate this problem in the future.

The biggest problem with CHEF's strategy is the need to search through previously encountered cases. Unless CHEF focuses on those case which are most relevant, CHEF's planning algorithm would be computationally intractable in a real world domain. This situation worsens as CHEF learns about an increasing number of cases. This is also true of CHEF's knowledge of plan failures. As CHEF learns about more plan failures, it

must search through these failures in order to determine if a plan will be successful in a particular problem situation.

While CHEF focuses on using plan failures in order to select plans, KIP focuses on the detection of these plan failures themselves. Such focusing would be very useful for finding potential plan failures in CHEF. As CHEF learns of more plan failures, the need to focus the planner's attention on the most likely plan failures becomes more acute. Concerns would also provide a better vocabulary for discussing the anticipation of plan failures.

2.2.3 SCRAPS

Hendler's [14] SCRAPS system addresses the problem of search through the use of a technique called *marker-passing*. Markers are passed from each concept to neighboring concepts in the network, and recursively on to other concepts. As markers are passed along they lose their *activation energy* and the marking stops. At this point, SCRAPS determines which concepts have the largest number of marks, i.e. collisions. In this way, SCRAPS can detect subplans that might result in interacting subgoals before SCRAPS attempts to use these subplans in its complete plans.

While marker-passing is an important search strategy for dealing with large bodies of knowledge, it has not been used in KIP. Instead, KIP's development has focused on knowledge structures, such as concerns, which are important for planning. However, such knowledge structures would be an useful addition to a planner which uses marker-passing. Markers could be passed through such knowledge structures causing collisions at aspects of a plan which should be considered.

Chapter 3

Representation of Planning Knowledge in KODIAK

3.1 Introduction

In this chapter, I discuss how plans and goals are represented in the KODIAK knowledge representation language. KODIAK is a semantic network knowledge representation language which has been developed by Robert Wilensky and other members of the Berkeley AI Research group. KODIAK concepts are represented as nodes connected together with primitive links. Wilensky[36] describes the motivations for the representation language and how it differs from other semantic network languages. This chapter focuses on the new ideas and new language features in KODIAK developed for use in planning. These ideas include a mechanism for describing equations between relation paths and a means of representing time points and time intervals.

KODIAK has two main components: (1) the KODIAK knowledge base and (2) the KODIAK interpreter. The knowledge base stores both long term memory regarding abstract concepts and a representation of objects in the world. The interpreter is used to retrieve facts from the knowledge base and to add facts about the world into the knowledge base. The implementation of KODIAK described is an extension of an implementation initially designed by Peter Norvig for the FAUSTUS natural language text inferencing system[30].

In this chapter, I first present an overview of the primitive links and concept types in KODIAK. I then focus on a particular type of structure in KODIAK, equations between relation paths. Equate relationships are particularly important for the specification of plans in KIP. Throughout the chapter I use examples of UNIX concepts represented in KODIAK. These concepts are used by the KIP planner in order to construct plans which will be suggested to the user.

3.2 Absolutes and Relations

There are two types of objects in KODIAK:

(1) *absolutes* - concepts that stand on their own, e.g. person, blue, command

(2) *relations* - relations between concepts, e.g. color-of, value-of

Absolutes refer to concepts that can be modeled in their own right. These include both physical objects and abstract ideas. In KIP, absolutes are used to represent concepts such as UNIX-COMMAND, FILE, and MACHINE.

Relations refer to objects that associate two concepts together. They are directional in that every relation points from a domain concept to a range concept. Examples of relations in KIP include File-Arg-Of-Unix-File-Command, File-Contents, and Command-Argument. (KODIAK objects are generally printed in small letters, with absolutes in all caps and relations capitalized.) Every relation also has an associated inverse relation that relates the range concept to the domain concept. (Such relations are usually printed with a *, e.g. File-Name*. Relations can only hold between two absolutes. Thus, a good working definition of the difference between absolutes and relations is that relations hold between absolutes, and absolutes are related by relations. All KODIAK relations are defined in the knowledge base. The exceptions are a few primitive links, known to the KODIAK interpreter, which are described in this chapter.

Both absolutes and relations can be further subdivided into two types of objects:

(1) categories - objects that refer to a class of objects

(2) individuals - objects that have extension in the world

For example, the concept FILE is an category absolute, which refers to the class of files. The File-Name relation is a category relation between a FILE and a FILE-NAME-STRING. This relation is represented in Figure 3.1 by the directed arrow from FILE to File-Name and from File-Name to FILE-NAME-STRING.

FILE-1 is an individual absolute which refers to a particular file in the world. (Individuals are always printed with a number on the end to indicate that they refer to a particular object.) Individual relations relate two individual absolutes. For example, if KIP knows that the name of the file FILE-1 is GEORGE-1, this information is stored in an individual relation File-Name-1 between FILE-1 and GEORGE-1. This relation is represented in Figure 3.2.

KIP's knowledge base includes information only about categories, but KIP's program uses KODIAK to construct individuals in the world about which KIP can reason. Unlike KL-ONE [5], individual objects are represented in the same way as categories. They

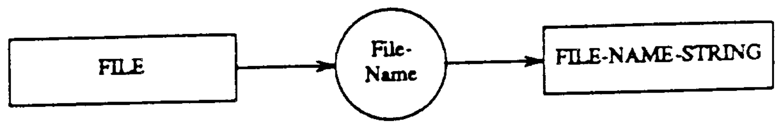

Figure 3.1: Representation of the File-Name relation

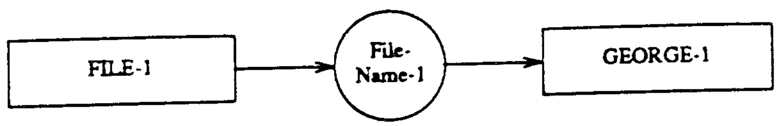

Figure 3.2: Representation of the File-Name-1 relation

are also implemented using many of the same LISP primitives. In the current implementation of KODIAK, there is a strict separation between categories and individuals in terms of relations. Category relations relate only category absolutes, and individual relations relate only individual absolutes.

3.3 Parent Links

KODIAK represents hierarchical relationships through the use of parent links. A parent link relates a parent concept to a child concept. The child concept inherits all of the relationships of the parent. For example, FILE is a parent of TEXT-FILE. Therefore, TEXT-FILE inherits all the relations of FILE. For example, the File-Name relation which is defined in Figure 3.1 is inherited by the TEXT-FILE absolute.

Parent links are further subdivided into two types of relations:

(1) dominate relations - parent links between two categories

(2) instance relations - parent links between a category and an individual

A dominate relation refers to a relationship in which one category is a subcategory of another category. An instance relation refers to a to a relationship in which a particular

object in the world is a of particular type. In KODIAK diagrams, dominate relations are represented by the letter 'D' and instance relations by the letter 'I'. For example, Figure 3.3 represents the fact that TEXT-FILE is dominated by FILE, and TEXT-FILE-1 is an instance of TEXT-FILE.

TEXT-FILE-1 is termed a *descendant* of FILE. Ancestor/descendant relations are defined recursively in terms of one more parent links.

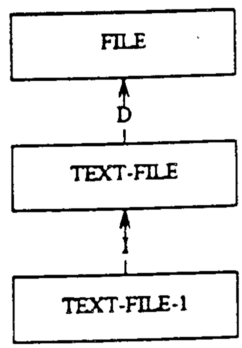

Figure 3.3: Parent Links

3.4 Relation Links: Domain and Range

Relations are related to absolutes by two types of links, domain and range. For example, in Figure 3.1, FILE is the domain of the File-Name relation and FILE-NAME-STRING is the range. The meaning of domain and range is depends upon on whether the relation is an individual or a category.

(1) categories relations - domain/range of relation is constrained to be an absolute

(2) individual relations - domain/range of relation is filled with an absolute

For example, in Figure 3.1 the range of the File-Name relation is *constrained* to be a FILE-NAME-STRING. This means that for any instance of the File-Name relation, the range must be a FILE-NAME-STRING. The domain of the File-Name relation is constrained to be a FILE. In Figure 3.2, the range of the individual relation File-Name-1 is *filled* with the

absolute GEORGE-1. Constraints are enforced by the KODIAK interpreter when the range of a relation is filled. Thus, if a program which uses KODIAK tries to fill in the range of an individual relation with an absolute that cannot satisfy the constraint, an error is signalled. In Figure 3.4, additional parent links are presented which demonstrate the constraints in representing the File-Name-1 relation. For example, GEORGE-1 must be an instance of FILE-NAME-STRING. Actually, GEORGE-1 need not be a child of FILE-NAME-STRING, but must be a descendant.

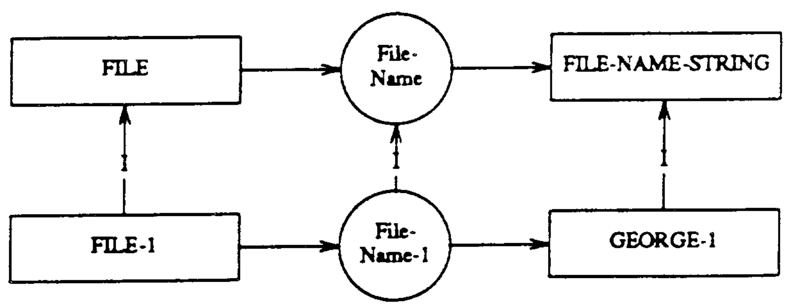

Figure 3.4: Parent Links of File-Name-1 Relation

3.5 Equate Links

Equate associations are KODIAK links which specify that the the ranges of two relations are always equal. Equates can be viewed as a substitute for variables in a representation and are an important part of the KODIAK representation language. When the range of a relation is filled in, any relations that are equated to such a relation must have the same range. This means that the range of both relations is the same absolute. Equations are defined in the KODIAK knowledge base between one relation and a *relation path*. A relation path is a composition of two or more relations.

For example, in Figure 3.5, there is an equation between the effected-file of the RM-COMMAND plan, and the relation path (Planfor Deleted-File). (An equate link is drawn between effected-file and the end of the relation path, i.e., Deleted-File. This means that the range of the effected-file relation must be the same as the range of the composition of the Planfor and Deleted-File relations. In other words, the effected-file of RM-COMMAND must be the same as the deleted-file of the goal for which the RM-COMMAND is a plan. It is important to represent these equations between a relation and a relation path rather than between two relations, since there might be many paths between any two relations.

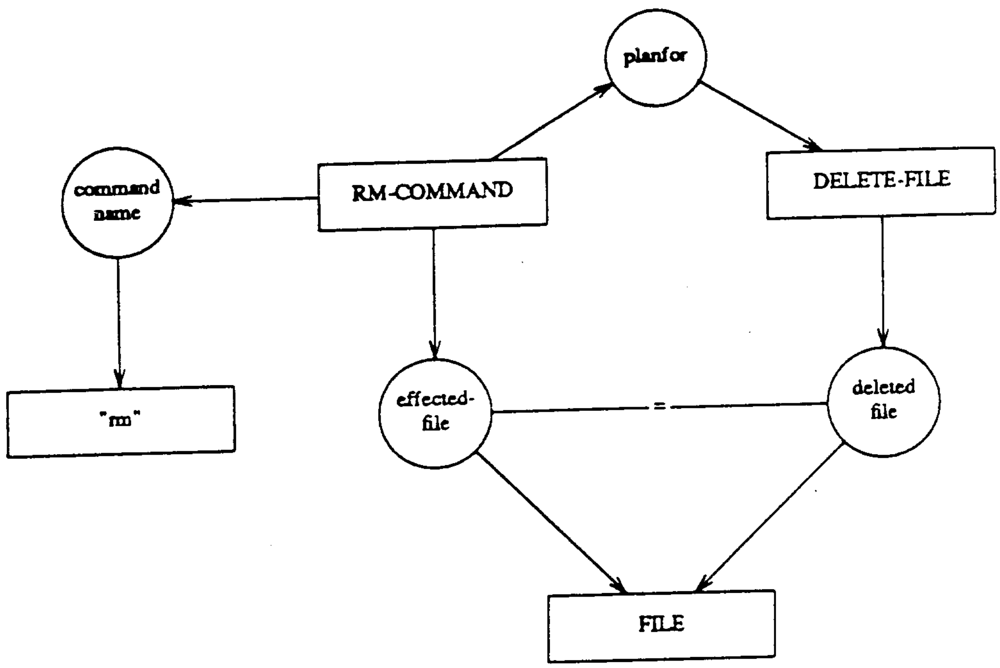

Figure 3.5: Use of Equates for RM-COMMAND Plan

KODIAK uses knowledge about equate relationships between relation categories in order to fill in the range of individual relations. The ranges of equated relations are filled in by KODIAK's *equate mechanism*. This mechanism uses information from the equate links in KODIAK's knowledge base to specify the domain/range of old relations and to create new relations. When KODIAK fills in the range of a equated relation, KODIAK ensures that the range of the relation is the same as the range of the relation at the end of the equated relation path. For example, consider the individual relations in Figure 3.6. In this

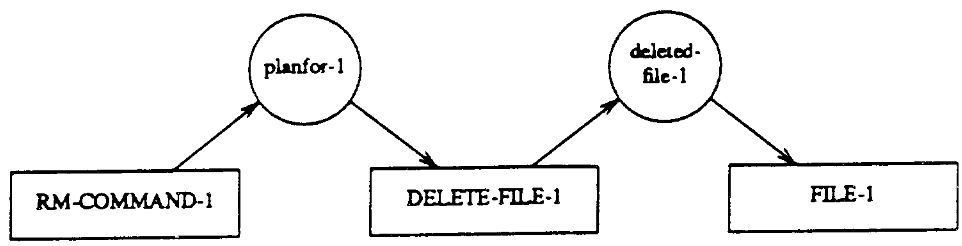

Figure 3.6: Existing Individual Relations of RM-COMMAND Plan

example, there exists an individual relation called Planfor-1 between RM-COMMAND-1 and DELETE-FILE-1, and an individual relation called Deleted-File-1 between DELETE-FILE-1 and FILE-1. Based on the equate knowledge in Figure 3.5, the equate mechanism creates an individual relation called Effected-File-1 between RM-COMMAND-1 and FILE-1. The Effected-File-1 relation is shown in Figure 3.7. A more complex example involving the equate mechanism is shown in Figure 3.10 on page 34.

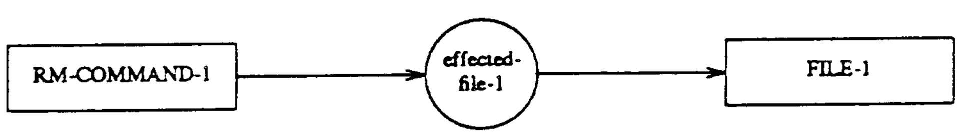

Figure 3.7: Individual Relation Created by Equate Mechanism

Brachman [5] describes structured associations of KL-ONE roles (which are similar to KODIAK relations). In one form, these structured associations can describe equality between what KL-ONE refers to as role chains. Role chains allow the knowledge base to encode role equality in a way similar to equations of relation paths. However, as discussed

in Section 3.2, individuals objects in KL-ONE are not represented in the same way as categories. Role chain knowledge must be duplicated in KL-ONE's A-BOX (Assertional Box), in order to be used in the instantiation of individual concepts. There is no way to coordinate these two types of knowledge. In fact, it is possible that equality between role chains might be contradicted in the assertional knowledge base. By using the same representation for categories and individuals, KODIAK is able to express equate knowledge about relation categories and use the equate knowledge to specify individual relations in the same form. Since KODIAK uses equate knowledge among relation categories to specify relationships between relation individuals, no contradictions between category equations and individual equations are possible.

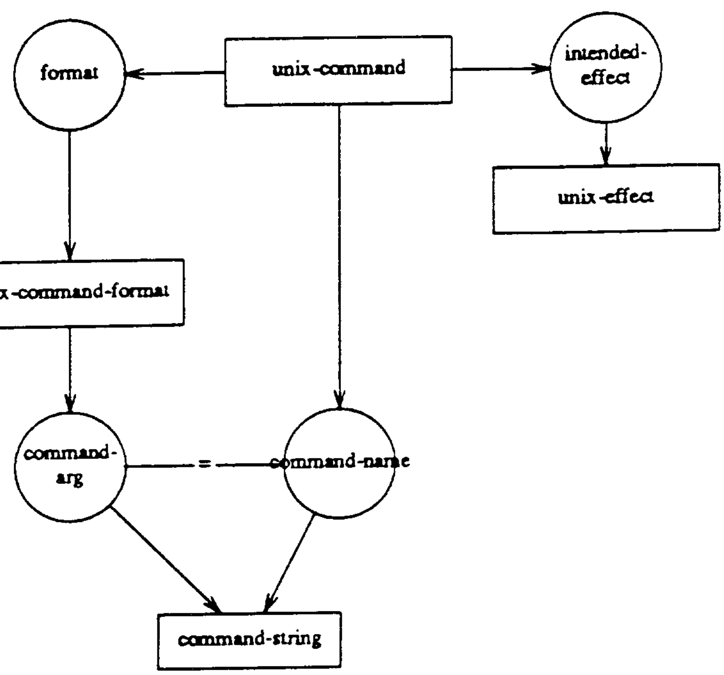

Figure 3.8: Use of Equates for UNIX-COMMAND Plan

Like other links in the KODIAK representation language, equates are hierarchical. One relation can inherit the equations of its ancestors in the hierarchy. For example, one of the ancestors of the RM-COMMAND plan is the category of UNIX-COMMAND. As illustrated in Figure 3.8, an equate link is made between the command-name relation and the relation path (Format Command-Arg). This means that the name of the command must be the same as the command argument of the command format. The RM-COMMAND plan in-

herits this relation from UNIX-COMMAND. Therefore, since it has been specified that the Command-Name of the RM-COMMAND plan is rm, rm must also be the command argument of its format. In order to invoke the command, a user types the name of the command. This is a general fact known about all UNIX commands.

A more specific category of UNIX-commands is UNIX-FILE-COMMAND. This is the category of UNIX-commands that operate on a file. The RM-COMMAND plan also inherits from this category. The format of UNIX-FILE-COMMAND is composed of a Command-Arg, (which was defined in Figure 3.8) and a Format-File-Arg, which is the name of the file. An example of such a format is "rm foo". As illustrated in Figure 3.9, there is an equate link between the File-Arg relation of UNIX-FILE-COMMAND and the relation path (Format Format-File-Arg File-Name*). This means that the filename of the File-Argument is the same as that of the Format-File-Arg. RM-COMMAND inherits from UNIX-FILE-COMMAND, and the effected-file of RM-COMMAND inherits from the File-Arg relation of UNIX-FILE-COMMAND.

During KIP's plan determination phase, KIP specifies a plan for the particular planning situation. Plan specification refers to specifying the values of the general plan for a particular problem situation. (Plan specification is described in detail in Section 6.2.1.) Once a plan has been fully specified, KIP can attempt to detect potential plan failures of the specified plan. In Figure 3.10, a KIP trace is presented that shows a specification of a RM-COMMAND plan, based on equate information represented in this section. A graphical representation of the individual RM-COMMAND created is shown in Figure 3.11.

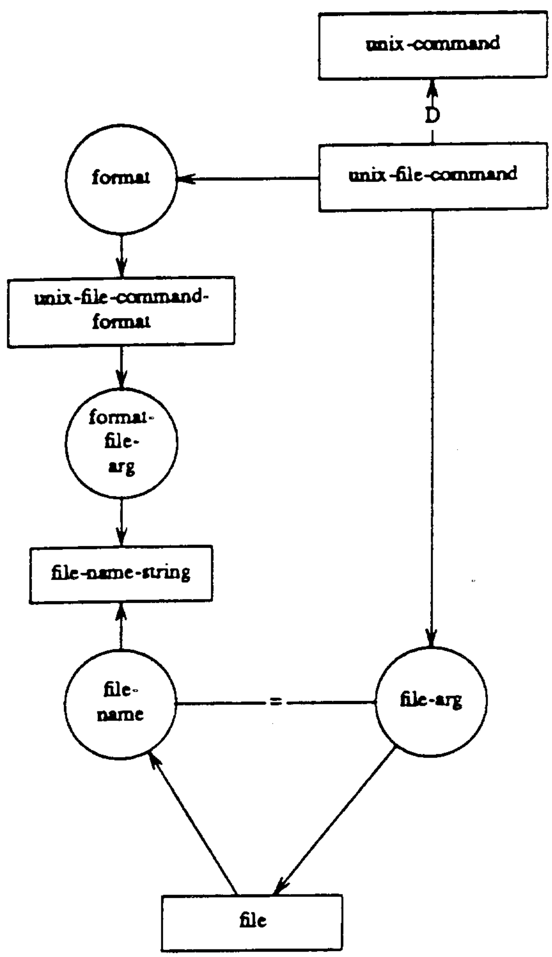

Figure 3.9: Use of Equates for UNIX-FILE-COMMAND Plan

Figure 3.10: KIP Trace of RM-COMMAND Construction Based on Equate Knowledge

User: How do I delete the file named dan?

PAGAN produces:
(delete-file-effect-1
 (Effected-File-23
  (file-1
   (File-Name-63 dan-3))))
;; *This is input KIP has received from the PAGAN goal analyzer. Based on the*
;; *values of the* Effected-File-23 *and* File-Name-63 *relations, and the equate*
;; *knowledge expressed in this section, KIP creates an entire plan.*

KIP is trying to determine a plan for the list of goals:
(delete-file-effect-1)
Selecting a goal from the List Of Goals ((delete-file-effect-1))
selecting the remaining goal
 delete-file-effect-1
Looking for a plan for the Current Goal (delete-file-effect-1)
First looking at stored plans
Selected the rm-command plan
Specifying the plan:
rm-command-50
(rm-command-50
 (File-Arg-Of-Unix-File-Command-53 file-1)
;; *KODIAK has specified a new instance of the rm-command plan,* rm-command-50.
;; *Based on the equate information from* rm-command *in Figure 3.5, KODIAK*
;; *has specified that the file-arg relation is* file-1 *which is the same as the*
;; *deleted-file of the* delete-file-effect-1.
 (Intended-Effect-Of-Rm-Command-51
  (delete-file-effect-1
   (Deleted-File-23
    (file-1
 (File-Name-63 dan-3)))))
 (Command-Name-Of-Rm-Command-57 rm-string-56)
 (Format-Of-Unix-Command-55
  (unix-command-format-54
   (Command-Arg-59 rm-string-56)
;; *KODIAK has specified the format of this particular rm-command based on*
;; *equates defined for* unix-command *in Figure 3.8.*
;; *In order to do this, KODIAK creates a new relation:* Format-Of-Unix-Command-55
;; *and fills it with a new absolute:* unix-command-format-54.
;; *KODIAK creates a new relation of* unix-command-format-54 *called* Command-Arg-59.
;; *This relation specifies the command argument that will be typed by the*
;; *user during plan execution. This relation is filled with the value*
;; rm-string-56. *This absolute represents the string "rm". The creation of this*

;; individual string is an artifact of the inability of the kodiak interpreter to
;; create relations between individual absolutes and category absolutes.

(Format-File-Arg-61 dan-3))))

;; KODIAK has specified the Format-File-Arg-61 *of the*
;; unix-file-command-format-54 *to be* dan-3 *based on equate knowledge regarding*
;; unix-file-command *in Figure 3.9. In this case, there was less additional*
;; processing necessary because the unix-command-format-54 *had already been set up*
;; while processing the equates of UNIX-COMMAND. In order to process this
;; equate relation, KODIAK had to follow more relations. The equation holds
;; between the File-Arg-Of-Unix-File-Command-53 *relation, and the relation path:*
;; (Format-Of-Unix-File-Command-55 Format-File-Arg-61 File-Name-63*). Thus, by*
;; using the equate relation stored on the file-arg relation, KODIAK followed
;; these three other relations. Of the three, only Format-File-Arg-61 *did not*
;; exist previously. This relation was filled with the file-name of file-1*:* dan-3.

;; At this point, KODIAK can present a plan to the user that he can actually
;; type in, i.e. rm dan.

3.6 Representation of Time

One of the most important concepts which need to be represented in KIP is state changes. Most goals in KIP's knowledge base are simple state changes. Since state changes represent changes in state over time, it is necessary to represent time relationships in KODIAK. Time is represented in both time intervals and time points. A particular time point is represented in terms of the points occurring earlier and later. Therefore, the set of time points which are stored in the KODIAK knowledge base is a partial ordering. Time intervals have a start time point and an end time point. Thus, my representation of time combines features of both Allen [1,2,3] and McDermott[27]. Allen's theory of time uses intervals (these are called periods in [3] to represent time relations while McDermott uses time points.

Relations that change over time can be said to be true during a particular time interval. For example, suppose that KIP knows that USER-1 has current-directory DIRECTORY-1 during a particular time-interval INTERVAL-1. During another time interval INTERVAL-2, USER-1 has current-directory DIRECTORY-2. These changes in the current-directory of USER-1 might be due to cd command which changes the current directory of the user. As shown in Figure 3.12, this is represented by the two relations Current-Directory-1 and Current-Directory-2.

The KODIAK interpreter is aware of a number of time interval relationships, including Before, After, Start-Before, Start-After. Since it is easier to represent relationships between points, KODIAK stores these time interval relationships by making assertions

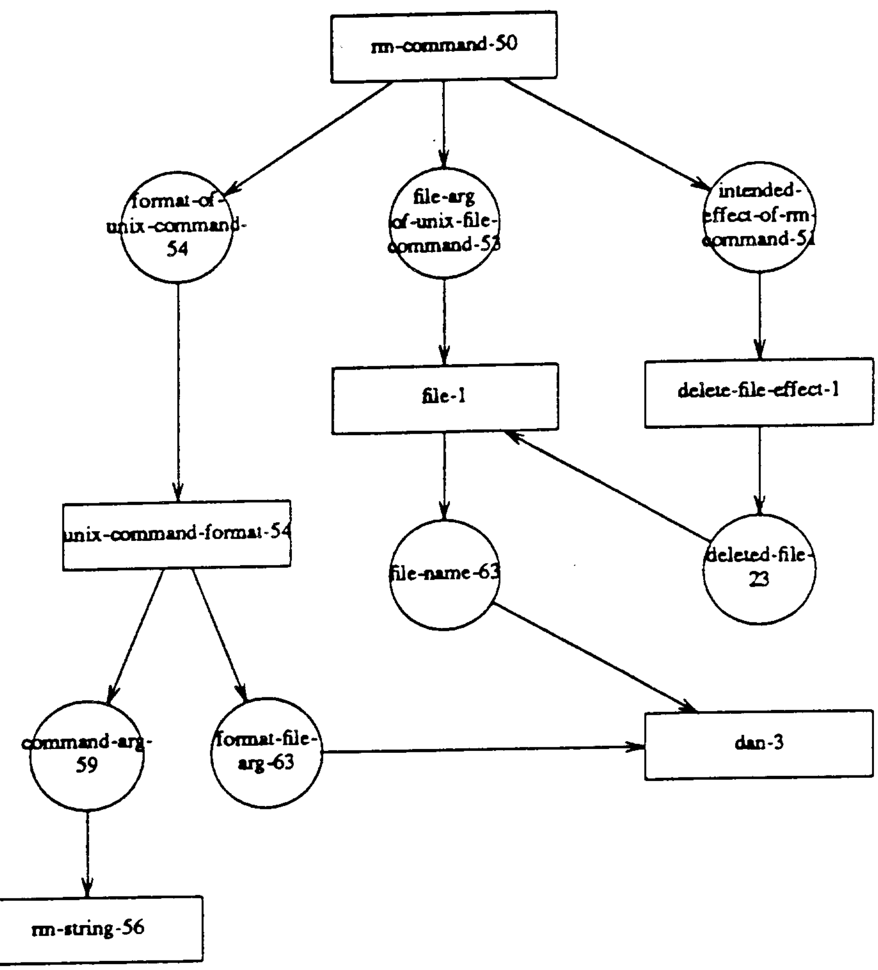

Figure 3.11: Representation of an Individual RM-COMMAND Plan

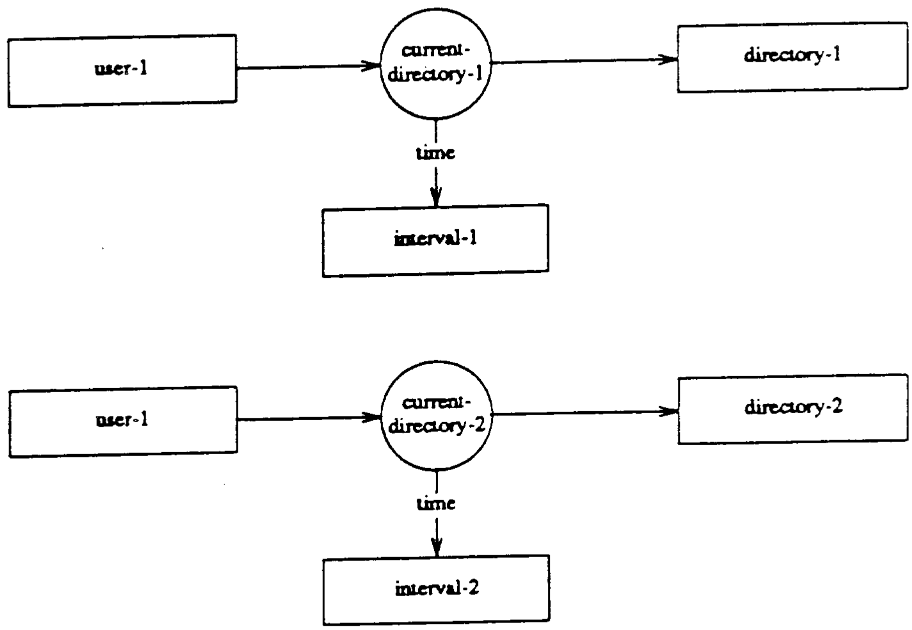

Figure 3.12: Representation of the Change of the Current Directory of USER-1

about their start and end points. For example, if Current-Directory-1 is before Current-Directory-2, then KODIAK asserts that the endpoint of INTERVAL-1 is earlier than the startpoint of INTERVAL-2.

These interval relationships can also be represented in KIP's long-term knowledge base by storing time relations between relations of the same absolute. For example, in order to represent the category of the state-change CHANGE-CURRENT-DIRECTORY, one must represent two Current-Directory relations. These relations are the initial state and final state of the particular state change. Furthermore, one must represent the fact that the initial state occurs before the final state. This fact is stored in the KODIAK knowledge base by a Before relationship between the initial state and final state. This relationship is illustrated in the representation of STATE-CHANGE shown in Figure 3.13.

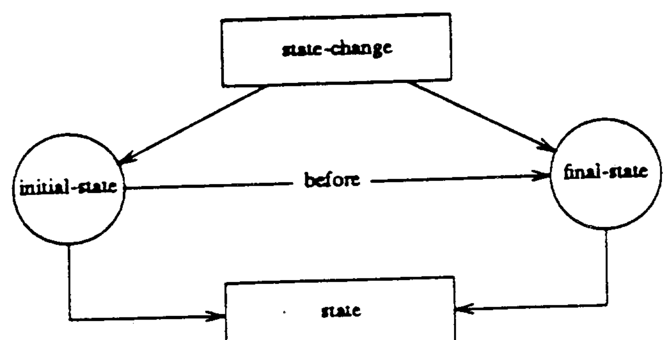

Figure 3.13: Representation of STATE-CHANGE

The representation of state changes is difficult for another reason. As noted in Section 3.2, relations can only relate two absolutes. However, the initial state and final- state relations described in the previous paragraph seem to be between a state-change absolute and a Current-Directory relation. Therefore, it is necessary to create a CURRENT-DIRECTORY-STATE absolute which is defined in terms of a Current-Directory relation. For example, as shown in Figure 3.14, a CURRENT-DIRECTORY-STATE-1 absolute is defined in terms of the Current-Directory-1 relation. Both CURRENT-DIRECTORY-STATE-1 and Current-Directory-1 share the same time-interval.

Reference time intervals are important to the representation of state-changes. They refer to the interval at which the state-change occurs. The initial state is true before the reference interval, while the final state is true after the reference interval. For example, in Figure 3.15, the initial CURRENT-DIRECTORY-STATE is true before the STATE-CHANGE-TIME and the final CURRENT-DIRECTORY-STATE is true after the STATE-CHANGE-TIME. During the reference interval, some other state(s) may be true.

Furthermore, representation of state changes include equate relationships which

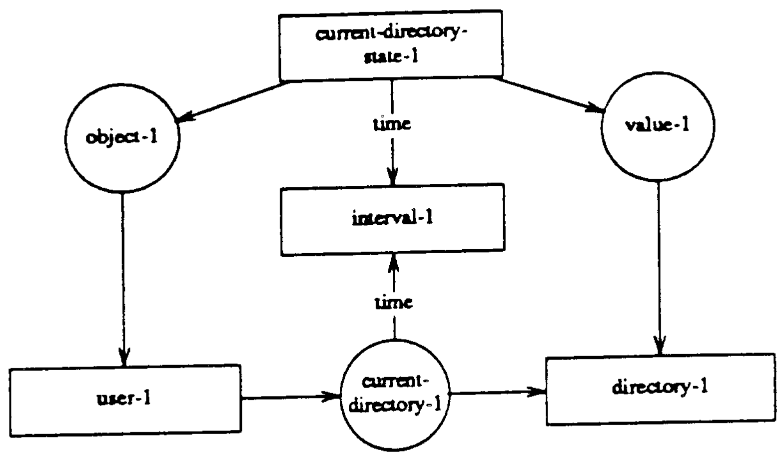

Figure 3.14: Representation of an Individual Current Directory State

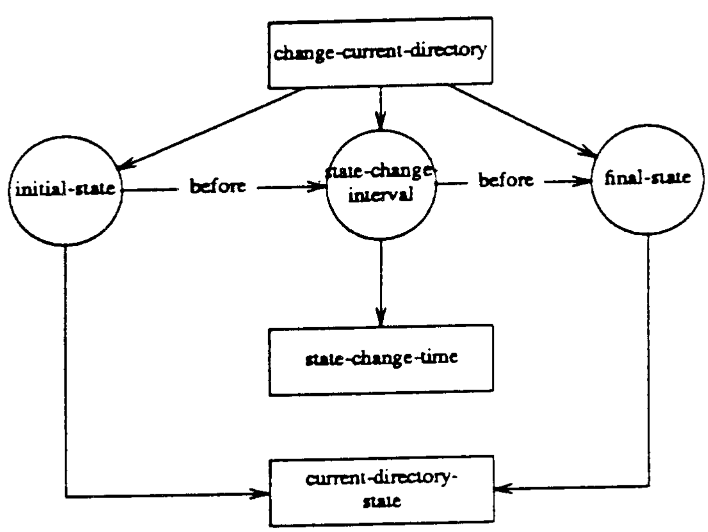

Figure 3.15: Representation of CHANGE-CURRENT-DIRECTORY

define new relations in terms of the initial and final states. In Figure 3.16, two equate relationships are defined. The Experiencer is equated to the relation path (Final-State Object). This means that the experiencer of the state change is the object of the final state. The Final-Value relation is equated to the relation path (Final-State Value). For example, when the Final-Value of the state-change is specified as DIRECTORY-3, a state created with the Experiencer of the state-change as the object and the DIRECTORY-3 as the value. A similar equate relationship is defined for the Initial-Value which is equated to the relation path (Initial-State Value). In addition, the Experiencer relation is also equated to the relation path (Initial-State Object). Experiencer's two equate relations encode the knowledge that the Experiencer is equal to both the object of the initial-state and the object of the final state.

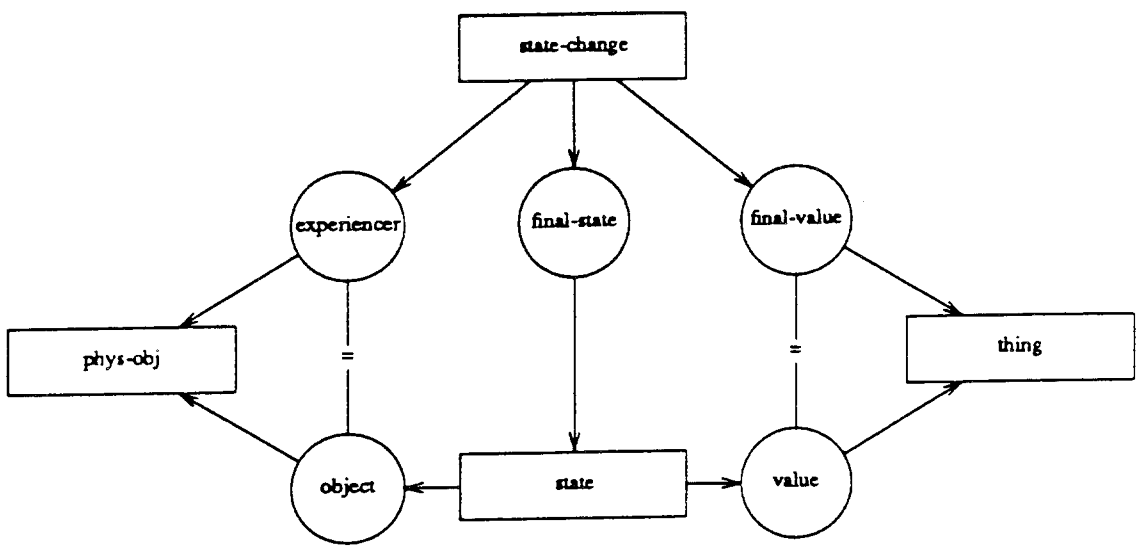

Figure 3.16: Equate Relationships of STATE-CHANGE

The creation of an individual CHANGE-CURRENT-DIRECTORY plan is presented in a KIP trace in Figure 3.17. This individual is created by specifying the Experiencer, Initial-Value, and Final-Value relations. In Figure 3.18, the individual CHANGE-CURRENT-DIRECTORY is represented in graphical form.

Figure 3.17: KIP-constructed Individual CHANGE-CURRENT-DIRECTORY State-Change

```
;; These are the commands that KIP sends to the KODIAK interpreter
;; The put-filler function is called with the arguments domain relation range
;; It creates an individual relation with the apropriate domain and range.
```

```
(put-filler CHANGE-CURRENT-DIRECTORY-1 EXPERIENCER USER-1)
(put-filler CHANGE-CURRENT-DIRECTORY-1 INITIAL-VALUE DIR-1)
(put-filler CHANGE-CURRENT-DIRECTORY-1 FINAL-VALUE DIR-2)

(change-current-directory-1
 (Experiencer-26 user-1)
 (Final-Value-39 dir-2)
 (Initial-Value-52 dir-1)
;; These are the values that are specified by KIP in the input to KODIAK
 (Final-State-Of-Change-Current-Directory-35
  (current-directory-state-27
   (Value-Of-Current-Directory-41 dir-2)
   (Object-Of-Current-Directory-31
    (user-1
     (Current-Directory-45 dir-1)
     (Current-Directory-30 dir-2)))))
;; This is the final-state of change-current-directory-1
;; The value is dir-2 as was specified above. This state was created due to equate
;; knowledge described in the previous figures
;; KODIAK prints these absolutes by printing all the relations of an absolute,
;; and then recursively printing each of the ranges of these relations. If the
;; absolute appears again in the same representation, the name of the
;; absolute is printed but not the relations. Thus, when KODIAK first prints
;; user-1, it prints all the relations of user-1. In this case, user-1 has two
;; current-directory relations, which are true at different time intervals.
 (Initial-State-Of-Change-Current-Directory-50
  (current-directory-state-42
   (Value-Of-Current-Directory-54 dir-1)
   (Object-Of-Current-Directory-46 user-1)))
;; This initial-state has the same object as that of the final-state, i.e. user-1,
;; but has a different value.
 (State-Change-Interval-37 state-change-time-36))
;; This is the reference time interval of the state-change. The interval-time
;; of current-directory-state-42 (the initial-state) is before the
;; state-change-time-36, and state-change-time-36 is before the interval-time of
;; current-directory-state-27 (the final-state). This means that the endpoint of
;; the interval-time of current-directory-state-42 is earlier than the start-point
;; of state-change-time-36, and the endpoint of state-change-time-36 is earlier
;; than the startpoint of the interval-time of current-directory-state-27.
```

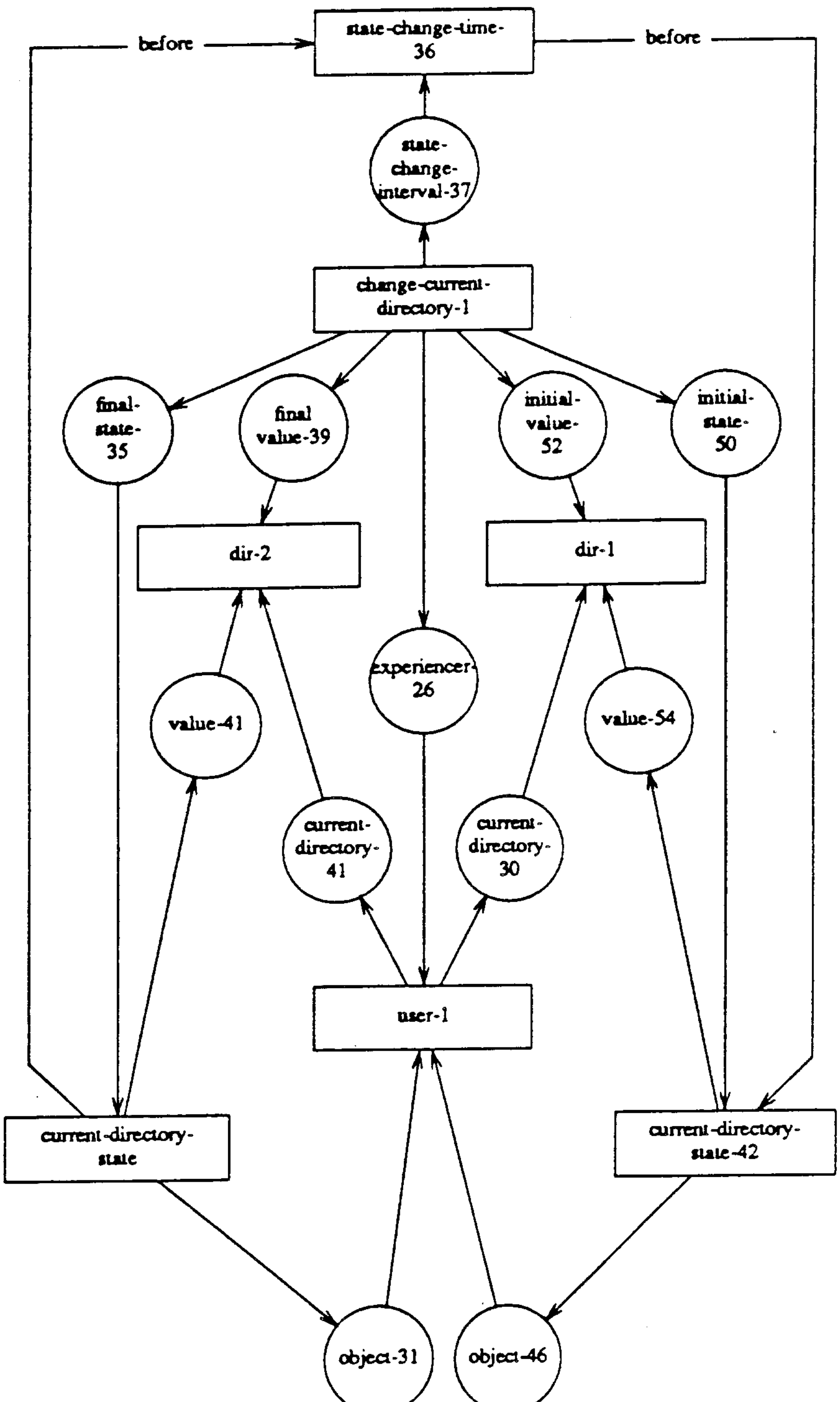

Figure 3.18: Representation of Individual CHANGE-CURRENT-DIRECTORY

3.7 Implementation of KODIAK

KODIAK is implemented in Common Lisp on a Symbolics Lisp Machine. Absolutes and relations are both stored as defstructs. Both share 5 slots:

(1) Name - name of the object

(2) Parent - list of the parents of the object

(3) Children - list of the children of the object

(4) Ancestors - list of all the ancestors of the object

(5) Time - time interval in which the object holds

The name slot's primary importance is in printing and debugging. The parent and children list include those objects directly related to the object through a parent relation. The ancestor list could be derived from the parent list. However, for efficiency and debugging considerations the complete ancestor list is maintained.

The time slot refers to a time interval. In the current implementation of KODIAK, time intervals are represented as separate defstruct objects. They have a start-point and an end-point which are both time-points. Time-points are implements are also implemented as defstruct objects. They have a list of earlier-points and a list of later-points. The time relationships that are described in this chapter are defined in the KODIAK interpreter. In future KODIAK implementations, these time object will be represented as KODIAK absolutes. Time relationships will be represented as KODIAK relations.

Absolutes have one additional slot:

(6) Relations - a list of relations of which this absolute is the domain

Relations have three additional slots:

(7) Range - the range of the relation

(8) Inverse- the inverse of the relation

(9) Equations- a list of relation paths which are equated to this relation

Inverse relations are usually given the name relation*. For example, the inverse of the File-Name relation is File-Name*.

A Relation paths is stored as a list of relations. For example, as shown in Figure 3.16 on page 40, the Final-Value relation is equated to the relation path (Final-State Value). Therefore, the Final-Value relation list stores the list (Final-State Value). Whenever, the range of a Final-Value relation is specified, KODIAK ensures that the relation at the end of the relation path has the same range. Presently, the equate mechanism uses an *always-fill-equates* strategy. When an individual equated relation is specified in, KODIAK fills in all the relation paths to which the relation is equated. If the relations in the relation path do not exist, KODIAK creates them. I am currently investigating ways of limiting the relations that are specified in until the range of the relation is needed.

3.8 Conclusion

In this chapter, I have described the KODIAK knowledge representation language. Two important features of the language have been discussed: (1) equations between relation paths, and (2) representation of time intervals and time points. In subsequent chapters, plans are described that use these KODIAK language features. For example, concerns depend heavily on the equate mechanism in order to evaluate a concern in a particular planning situation.

Chapter 4

Overview of Plan Synthesis in KIP

4.1 Introduction

In this chapter, I present an overview of the process used by KIP to synthesize a plan which will satisfy the goals of the user. When KIP is asked to provide a complete plan for a particular problem situation, it identifies the goals that need to be addressed, formulates a plan for these goals, and determines if the plan will work in the current planning situation. If the plan doesn't work, the plan is modified or an entirely new plan is constructed. The process of constructing a plan for the user's goals is called *plan synthesis*.

Plan synthesis is an iterative process. KIP decides which of the user's goals it will try to satisfy initially. As described in the next section, most of these goals come from the PAGAN goal analyzer[25]. KIP then attempts to determine a plan for this goal. KIP tests the plan to determine whether it will really satisfy the user goal in this particular planning situation. The two ways that a plan may fail are termed condition failure and goal conflict failure. If a certain condition needs to be satisfied in order to execute the plan, the satisfaction of the condition becomes a new goal that needs to be satisfied. If the determined plan causes a conflict with another user goal, the resolution of the goal conflict becomes a new goal. Finally, KIP needs to determine if all the user goals have been satisfied. If all user goals are not satisfied, KIP iterates through the process again. In this way, a plan is gradually synthesized. KIP's plan synthesis algorithm is outlined in Figure 4.1 below.

This chapter begins with a brief discussion of the three plan modules of this interactive process:

(1) *goal establishment* -	establishment of those goals KIP needs to address
(2) *plan determination* -	determination of a plan for those goals
(3) *plan failure detection* -	detection of potential plan failures

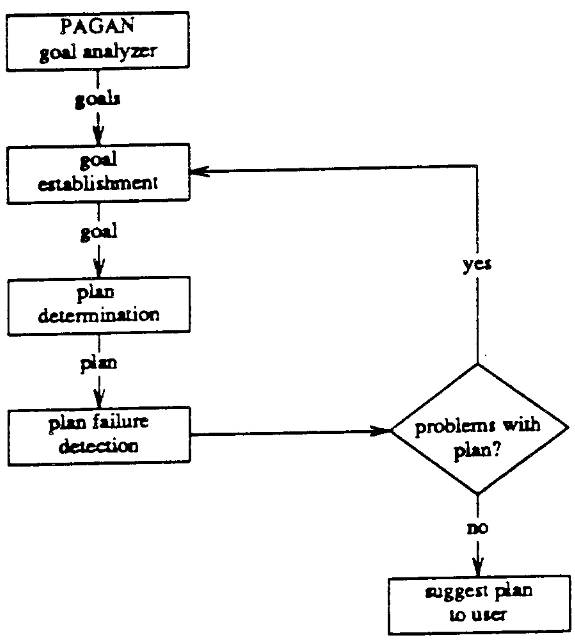

Figure 4.1: Outline of Plan Synthesis

In the remainder of this chapter, the entire plan synthesis process is outlined through a description of the three plan modules' use in plan synthesis to synthesize a plan. Goal establishment and plan determination will be discussed more fully in Chapters 5 and 6. Plan failure detection, the focus of this thesis, will be discussed in Chapters 7 through 10.

4.2 Plan Modules of KIP's Planning Algorithm

In this section, each of three plan modules of KIP's planning algorithm is described. The following section, containing a description of the entire plan synthesis algorithm, describes the way in which the three plan modules interact to synthesize a complete plan.

4.2.1 Goal Establishment

During *goal establishment*, KIP decides which goals should be considered during an iteration of the plan synthesis. There are two parts of this process:

(1) *Goal Detection* - detection of goals for the planning situation

(2) *Goal Selection* - selection of one goal from the many goals that need to be satisfied

Goal Detection is further decomposed into 3 parts corresponding to three types of goals that are detected:

(a) *Expressed goal detection* - goals in the user's utterance

(b) *Inferred goal detection* - goals inferred from the planning situation itself

(c) *Interest-derived goal detection* - goals which reflect threatened user interests

Expressed goal detection refers to KIP's initial detection of its goals through the examination of the result from the PAGAN goal analyzer. Inferred goal detection refers to the detection of goals during subsequent iterations as a result of failure of KIP's own plans to address these goals. KIP also determines if user *interests* should be detected to reflect goals in the present planning situation. Interests are general states that KIP assumes are important to the user. When interests tbecome active, they can give rise to one or more goals. When an interest is threatened, it gives rise to goals that will prevent that interest from being threatened. Interests are discussed further in Chapter 5.

If KIP has more than one goal to address, it selects one of the goals during goal selection. KIP tries to select the goal which is most important. The importance criteria will be discussed more fully in Chapter 5.

4.2.2 Plan Determination

Plan determination refers to the process of determining a plan for the goal which has been selected. There are two parts of plan determination:

(1) *Plan Selection* -	selecting a plan for a user goal
(2) *Plan Specification* -	specifying the values of a general plan for a specific planning situation

During plan selection, KIP first tries to select a plan which has been previously stored as the plan for the current goal. KIP assumes that the stored plan will be better than any new plan it could devise. KIP finds a stored plan for a particular goal, by looking at stored plans for a parent of the goal instance. The parent goal organizes information about the particular goal, without specifying certain values of the goal. For example, the name of a file which is the object of a goal instance would not be specified in the parent goal. If there is more than one plan for this goal, KIP must choose among alternative plans. If there is no stored plan that addresses the goals of the user, KIP must find a new plan. If no such plan exists, KIP tries to modify another plan to work in the current planning situation.

During plan specification, KIP specifies values for the particular planning situation that have been unspecified in the general plan description. Suppose the user asks the following question:

(1) `How do I list all the files in the directory named koala?`

In (1), during plan selection, KIP selects the stored plan for the goal of listing the contents of a particular directory, the USE-LS-COMMAND plan.

During plan specification, KIP creates an instance of this plan, called the USE-LS-COMMAND1 plan. KIP uses the plan description of the USE-LS-COMMAND plan to specify the values of the directory name argument for use in the ls command. The value of the directory name argument is specified in the plan description as always being equal to the name of the directory in the LIST-ALL-FILES-IN-DIRECTORY goal. This equality is represented by using an equate link in the KODIAK network. A detailed description of equate links and KIP's plan determination algorithm will be presented in Chapter 6.

4.2.3 Plan Failure Detection

Once KIP has determined a potential plan for a goal, it must determine if the plan will work in the present planning situation. This process is termed *plan failure detection* . KIP must detect two kinds of plan failures:

(1) condition failure - failure due to an unsatisfied condition necessary for proper plan execution

(2) goal conflict failure - failure due to an effect of the plan conflicting with another user goal

These failures are detected by using a knowledge structure termed a *concern*. Concerns identify which aspects of a plan are most likely to cause plan failure. They are added to the knowledge base by the planner knowledge base designer to reflect his experience of plan failures. There are two different types of concerns that address the two types of plan failures:

(1) *Condition Concerns* - concerns about the conditions of a plan which are most likely to be unsatisfied

(2) *Goal Conflict Concerns* - concerns about potential conflicts between effects of a plan and a user goal

In example (1), during plan failure detection, KIP examines the condition concerns for the USE-LS-COMMAND . In this way, KIP determines if new conditions need to be satisfied in order for the USE-LS-COMMAND1 plan to execute successfully. In this particular case, all of the many conditions for the USE-LS-COMMAND are likely to be met. Thus, there are no concerns involved.

However, if the user had asked the following question:

(2) `How do I list all the files in Jim's directory named koala?`

KIP would have to consider a concern regarding the read permission of Jim's directory. The USE-LS-COMMAND plan will not work if the user does not have read permission on the directory the user wishes to list. The read permission condition is typically a cause for concern when accessing other users' files. The method in which such concerns are accessed in example (2), will be explained in Chapter 8.

In example (1), since KIP has not detected any condition failures during the plan failure detection phase, no new goals for the satisfaction of conditions are added to the

set of goals for which KIP is planning. In addition, no goal conflict concerns are listed for the USE-LS-COMMAND plan. Therefore, no goal conflict failures are detected and no goal conflict resolution goals are added to the set of KIP's goals. Since KIP's only goal is satisfied, the USE-LS-COMMAND1 plan can be passed to UCEGO mechanism so that the plan can be suggested to the user.

4.3 Plan Synthesis Algorithm Description

In this section, I describe the way in which the plan synthesis modules described in the previous section work together to form a complete plan which can be suggested to the user.

Most plans cannot be synthesized in a single pass for one or both of the following reasons:

(1) the potential plan does not satisfy all the goals of the user

(2) KIP has concerns about the potential plan

KIP treats both these cases symmetrically. KIP generates new goals that reflect the need to dispose of concerns about the potential plan. KIP then adds these *concern-disposal* goals to the set of unsatisfied goals. KIP then reiterates the planning algorithm starting with this new set of goals. This process repeats itself until either KIP decides that it has determined a satisfactory plan or KIP decides to try a new plan for the user's goals. The determination of a satisfactory plan occurs when no user goals or concerns about the potential plan remain. KIP decides to try a new plan when one of the user goals cannot be satisfied or the concerns caused by the potential plan cannot be avoided. If necessary, KIP backtracks completely to its first plan selection. Backtracking[10],fikes is seldom a major efficiency problem for KIP because the plans that it synthesizes are constructed from a small number of known plans. Issues relating to KIP's plan selection algorithm are discussed further in Chapter 6.

4.4 Plan Synthesis Example

In this section, an example of plan synthesis is presented. Due to the interaction between plan modules, it is necessary for KIP to iterate through each of these twice. These plan modules will discussed in the order that KIP addresses them.

Suppose the user asks the following question:

(3) `How do I print the file eric on the apple printer?`

4.4.1 First Iteration of Planning Synthesis

Goal Establishment KIP's initial goal detection from the planning situation is the input of goals from the PAGAN goal analyzer. PAGAN creates an individual goal which is dominated by the PRINT-FILE-ON-APPLE-PRINTER goal. Let us call this individual goal PRINT-FILE-ON-APPLE-PRINTER1. PAGAN fills in the specific value of the file to be printed, as the file named eric. Since the user has specified that he wants to print the file on *the printer*, PAGAN tries to resolve the printer reference. PAGAN assumes that the apple printer referred to by the user is the printer in his office, because PAGAN believes that users generally refer to objects which are the easiest to access. PAGAN uses default knowledge in order to resolve references in the user's expressed goal. KIP assumes that defaults are true, unless KIP determines that a default is violated in a particular planning situation. Defaults and violated defaults are discussed in detail in Section 8.5.

In this phase of this example, there is no need to instantiate any goals to reflect user interests. Goals due to interests are not usually instantiated during the first iteration of interest-derived goal detection. Interest detection occurs only in response to a potential plan failure, and KIP has not yet selected a potential plan. KIP is usually not considering any other active interests. In most cases, no interests are expressed by the user in his question. Interests expressed in previous utterances of the current conversation are usually no longer active, since KIP tries to address these interests by suggesting plans that will not threaten them.

At this point, KIP is considering only one active goal. Therefore, there is no need to select among competing goals during goal selection.

Plan Determination KIP first tries to find a stored plan for the individual goal, PRINT-FILE-ON-APPLE-PRINTER1. There is no stored plan for this very specific goal. There is only one plan for the general goal of PRINT-FILE-ON-APPLE-PRINTER, the USE-LPR-AP-COMMAND plan. Therefore, KIP selects USE-LPR-AP-COMMAND as a potential plan for the PRINT-FILE-ON-APPLE-PRINTER1 goal. KIP then creates a USE-LPR-AP-COMMAND1 plan for this particular planning situation. The individual plan is then specified during the plan specification phase. The values which must be specified include the name of file to be printed and the location of the particular apple printer in the user's office.

Plan Failure Detection KIP next tries to detect any plan failures that might result from executing the USE-LPR-AP-COMMAND1 in the current planning situation. KIP first tries to detect any likely condition failures. KIP thus examines the condition concerns of the USE-LPR-AP-COMMAND1 plan. There is one concern for this plan that KIP considers. This concern encodes the possibility that the printer may run out of paper.

This concern is based on the experience of the planner knowledge base designer. It reflects the experience that this type of printer has a small paper tray and runs out of paper

fairly often. If the printer has no paper, the user's goal of PRINT-FILE-ON-APPLE-PRINTER1 will not be satisfied. KIP evaluates this concern in the particular planning situation and decides that this concern is a potential source of plan failure. (The method KIP uses to make this decision will be described in Chapter 8.) Therefore, a new goal is added to KIP's list of goals that must be satisfied, namely, having paper in the printer. Let us call this goal HAVE-FULL-PAPER-TRAY.

KIP next attempts to detect plan failures due to goal conflict. In this case, there are no goal conflict concerns for the USE-LPR-AP-COMMAND1 plan. Therefore, no plan failures due to goal conflict are detected and no goals to resolve goal conflicts are added to KIP's list of goals that must be satisfied.

4.4.2 Second Iteration of Plan Synthesis

Goal Establishment At this point in the plan synthesis process, KIP has determined a plan for the user's goal. KIP has determined that the plan will most likely satisfy the goal except for one concern. This concern has given rise to a new goal, the HAVE-FULL-PAPER-TRAY goal.

Since there are no potential goal conflicts, no interests of the user have been threatened. Also, since there is only one goal that needs to be satisfied, no goal selection is necessary. Thus, the HAVE-FULL-PAPER-TRAY goal is passed on to the plan selection process.

Plan Determination KIP thus attempts to find a plan for the goal of having a full paper tray in the printer. There is a stored plan for this goal, the FILL-PAPER-TRAY plan. This plan entails the user walking over to the printer, check the paper, and if necessary fill the paper tray.

KIP creates an instance of this plan and specifies the values of the plan during plan specification. KIP specifies that the particular printer must be filled with paper. Furthermore, KIP decides that the FILL-PAPER-TRAY1 subplan is added before the USE-LPR-AP-COMMAND1 subplan. This is because the FILL-PAPER-TRAY1 subplan was designed to satisfy conditions needed for the USE-LPR-AP-COMMAND1 subplan to execute successfully.

Plan Failure Detection KIP is faced with two related decisions at this point in the plan failure detection process. KIP must decide if the subplan chosen for the HAVE-FULL-PAPER-TRAY1 goal will work in this particular planning situation. KIP must also decide if any of the effects of this subplan conflict with the total plan developed through this point in the planning situation process. Particularly, KIP must determine if the current subplan might either delete conditions necessary for the execution of another subplan, or conflict with one of the desired effects of another subplan.

KIP addresses both these issues by examining the concerns of the FILL-PAPER-TRAY1. In the normal situation, there are no concerns for this plan. This plan is relatively easy to satisfy, and has no likely side effects. Also, since few plans will benefit from an empty paper tray, this plan is very unlikely to give rise to interacting subgoals. Therefore, the FILL-PAPER-TRAY1 plan has no interacting subgoal concerns.

Since there are no more additional goals or concerns to be addressed, the plan can be passed back to the UCEgo mechanism in order to be suggested to the user.

4.5 A KIP Trace Example

In this example, I present a KIP trace of plan synthesis. In this example, KIP must iterate through its planning steps three times.

Figure 4.2: KIP Trace of Plan Synthesis

How do I print the file named secret on the office printer?

;; KIP receives the following goal as input from the parser and the PAGAN goal
;; analyzer:

KIP is trying to determine a plan for the list of goals:

```
(print-file-effect-1
 (Destination-Of-Office-Printer-Print-File-Effect-25 office-printer-1)
 ;; KIP has received the goal of printing a file on the office printer.
 (File-Arg-Of-Print-File-Effect-30
  (file-1
   (Contents-34
    (file-contents-33
     (Printing-Of-36 printing-31)))
   (File-Name-29 secret-1)))
 ;; KIP knows that the file has some contents, and the contents has a
 ;;printing-of, i.e. printing-31. printing-31 is the object that appears on
 ;;the paper. Furthermore, the name of the file is the name secret.
 (Output-49
  (paper-38
   (Printed-On-53 blank-63)
   (Printed-On-40 printing-31)))
 ;; The experiencer of this state-change is paper-38.
 ;; It has two printed-on relations. Before the state-change, the paper is blank.
 ;; After the state-change, the paper has printing-31 on it.
 (Output-Printed-On-32 printing-31)
 (Initial-Value-Of-Print-File-Effect-60 blank-63)
 (Final-State-Of-Print-File-Effect-45
```

```
(printed-on-state-37
  (Value-Of-Printed-On-65 printing-31)
  (Object-Of-Printed-On-41
   (paper-38
    (Printed-On-53 blank-63)
    (Printed-On-40 printing-31)))))
 (Initial-State-Of-Print-File-Effect-58
  (printed-on-state-50
   (Value-Of-Printed-On-62 blank-63)
   (Object-Of-Printed-On-54 paper-38))))
```

;; *The final state and initial state refer to the state of the paper before and after*
;; *the state-change.*

Entering Goal Establishment Phase:

Selecting a goal from the List Of Goals ((print-file-effect-1))
selecting the remaining goal
print-file-effect-1

;; *Since KIP has only one goal, there is no need to select among goals*

Entering Plan Determination Phase:

Looking for a plan for the Current Goal (print-file-effect-1)

First looking at stored plans

Selected lpr-pop-command as a potential plan

Now specifying the plan for the particular planning situation:

```
(lpr-pop-command-66
 (Destination-Of-Lpr-Command-69
  (office-printer-1
   (Printer-Abbreviation-Of-Office-Printer-92 op-string-94)))
 (Format-Of-Lpr-Command-75
  (unix-file-printing-command-format-74
   (Command-Arg-89 lpr-string-86)
   (Printer-Arg-91 op-string-94)
   (Format-File-Arg-83 secret-1)))
```

;; *The user could execute this plan by typing lpr -Pop secret*

(Intended-Effect-Of-Lpr-Pop-Command-67 print-file-effect-1)

;; *The representation of* print-file-effect-1 *is shown above.*

(Actor-Of-Unix-Command-99 uc-user-1)
(File-Arg-Of-Unix-File-Command-77 file-1))

;; file-1 *is the file which the user wants to be printed.*

Entering Plan Failure Detection Phase:

;; *KIP is now searching for concerns of the* lpr-pop-command *plan*

Evaluating the condition concern: office-printer-room-door-locked-109

;; *This concern reflects a potential plan failure of the* lpr-pop-command. *Since the*
;; *office printer room is always locked, the user must have a key to the room in*
;; *order to pick up his output.*

;; KIP now evaluates this concern in this particular planning situation:
The condition of concern is the has-key-state-121 of the
uc-user-1
The current value of the condition is key-123
The desired value is key-111

;; KIP is evaluating whether the user has a key to the office printer room, i.e.
;; key-111. *Unfortunately, the user has* key-123. *This is actually KIP's way of*
;; representing a placeholder for the concept key. In other words, the user has
;; some key, but it is not key-111.

Instantiate concern that current value be changed
Creating a goal that reflects a change from the Current Value (key-123)
to the Desired Value (key-111)

;; KIP is now instantiating a goal to reflect the office-printer-room-door-locked-109 *concern*
Creating the goal:

```
(have-office-key-effect-201
 (Final-Value-Of-Have-Office-Key-Effect-204 key-111)
 (Initial-Value-Of-Normal-State-Change-176 key-123)
 (Final-State-Of-Have-Office-Key-Effect-219
  (has-key-state-216
   (Object-Of-Has-Key-217 uc-user-1)
   (Value-of-Has-Key-221 key-111)))
 (Initial-State-Of-Have-Office-Key-Effect-202
  (has-key-state-121
   (Value-Of-Has-Key-135 key-123)
   (Object-Of-Has-Key-125 uc-user-1))))
```

;; In the initial state, the user has key key-123.
;; In the final state, the user has key key-111.

Asserting the fact that the final-state of the goal (has-key-state-216)
starts before the start of plan interval (lpr-pop-command-66)
So that the condition holds before the plan is executed

;; Before constructing a plan for the have-office-key-effect-201 *goal, KIP first*
;; evaluates one more concern in the particular planning situation.

Evaluating the condition concern: out-of-paper-concern-136

;; This is the concern that the printer is likely to run out of paper.

;; KIP evaluates this concern by determining how much paper in is the paper tray.

The condition of concern is the paper-tray-status-of-office-printer-state-164 of the
office-printer-1
The current value of the condition is empty-or-full-142
The desired value is full
Current Value (empty-or-full-142) is less specific than the Desired Value (full)

;; KIP does not have information about the paper tray of this printer
;; Therefore, KIP consults it default mechanism

Try to make the current value more specific using defaults
Default-value is empty.
The Default Value (empty) is mutually exclusive with respect to the Desired Value (full)

Since the Default Value (empty) is more specific than the Current Value (empty-or-full-142), instantiate concern that default value be changed

Creating a goal that reflects a change from the Current Value (empty) to the Desired Value (full)

;; KIP is creating the goal of filling the paper tray with paper.

Creating the goal:

```
(have-full-paper-tray-effect-249
 (Final-Value-Of-Have-Full-Paper-Tray-Effect-267 full-270)
 (Initial-Value-Of-Normal-State-Change-250 empty-286)
 (Final-State-Of-Have-Full-Paper-Tray-Effect-265
  (paper-tray-status-of-office-printer-state-282
   (Value-Of-Paper-Tray-Status-Of-Office-Printer-293 full-270)
   (Object-Of-Paper-Tray-Status-Of-Office-Printer-278
    (office-printer-1
     (Printer-Room-Of-Office-Printer-114
      (main-office-113
       (Door-Of-116
        (door-115
         (Key-Of-118 key-111)))))
     (Paper-Tray-Status-Of-Office-Printer-277 full-270)
     (Paper-Tray-Status-Of-Office-Printer-237 empty-286)))))
 (Initial-State-Of-Have-Full-Paper-Tray-Effect-247
  (paper-tray-status-of-office-printer-state-234
   (Value-Of-Paper-Tray-Status-Of-Office-Printer-254 empty-286)
   (Object-Of-Paper-Tray-Status-Of-Office-Printer-238 office-printer-1)))
 (Experiencer-Of-Have-Full-Paper-Tray-Effect-256 office-printer-1)
 (State-Change-Interval-244 state-change-time-243))
```

Asserting the fact that the final-state of the goal
(paper-tray-status-of-office-printer-state-282) starts before the
start of plan interval (lpr-pop-command-66)
So that the condition holds before the plan is executed

Entering Goal Establishment Phase:

;; This is the beginning of KIP's second iteration of plan synthesis.
;; In this iteration, KIP is addressing goals that were generated
;; to reflect concerns that KIP has about the lpr-pop-command-66 *plan.*

Selecting a goal from the List Of Goals
((have-full-paper-tray-effect-249 have-office-key-effect-201))
Sorting the goals according to importance level
The List Of Goals ((have-full-paper-tray-effect-249 have-office-key-effect-201))
is already sorted in order of importance

Entering Plan Determination Phase:
Looking for a plan for the Current Goal (have-full-paper-tray-effect-249)

First looking at stored plans
Selected fill-paper-tray as a potential plan
Now specifying the plan for the particular planning situation:

;; KIP is now specifying the plan regarding filling the office printer tray.

;; *During the specification process, the* fill-paper-tray *plan instance is concreted to*
;; *an instance of* fill-office-printer-paper-tray.

```
(fill-office-printer-paper-tray-307
 (Fill-Paper-Tray-Printer-Tray-Of-Fill-Office-Printer-Paper-Tray-305
  (office-printer-1
   (Printer-Room-Of-Office-Printer-114
    (main-office-113
     (Door-Of-116
      (door-115
       (Key-Of-118 key-111)))))
   (Paper-Tray-Status-Of-Office-Printer-277 full-270)
   (Paper-Tray-Status-Of-Office-Printer-237 empty-286)
   (Printer-Abbreviation-Of-Office-Printer-92 op-string-94)))
 (Machine-Of-Of-Unix-Command-318 machine-100)
 (Intended-Effect-Of-Fill-Paper-Tray-302 have-full-paper-tray-effect-249 )
 (Actor-Of-Unix-Command-313 uc-user-1))
```

Asserting that fill-office-printer-paper-tray-307 comes before lpr-pop-command-66

Entering Plan Failure Detection Phase:
No plan failures are detected

Entering Goal Establishment Phase:

;; *This is the beginning of KIP's third iteration of plan synthesis.*
;; *In this iteration, KIP is addressing the remaining concern-generated goal.*

Selecting a goal from the List Of Goals ((have-office-key-effect-201))
selecting the remaining goal

Entering Plan Determination Phase:
Looking for a plan for the Current Goal (have-office-key-effect-201)

;; *The representation of* have-office-key-effect-201 *is shown above on*
;; *page 55.*

First looking at stored plans

Selected get-key-from-office as a potential plan

Now specifying the plan for the particular planning situation:

```
(get-key-from-office-408
 (Intended-Effect-Of-Get-Key-From-Office-409 have-office-key-effect-201 )
 (Actor-Of-Get-Key-From-Office-411 uc-user-1))
```

Asserting that get-key-from-office-408 comes before lpr-pop-command-66

Entering Plan Failure Detection Phase:
No plan failures detected for Candidate Plan (get-key-from-office-408)

To print the file named secret on the office printer, type lpr -Pop secret.
But first, get the key to the office printer room and fill the printer with paper.

4.6 KIP Implementation

KIP is implemented in Common Lisp on a Symbolics 3670. It depends heavily on the KODIAK knowledge representation language interpreter. KIP's includes over 1000 absolutes and relations which represent approximately 50 plans, 50 effects, and 25 concerns. Many concerns apply to multiple plans. Unlike previous UC implementations, multiple inheritance is pervasive. Many concepts have 3 or more parents.

Running times range between .5 seconds for simple examples to 10 seconds for more complex examples. As much as 80% of the processing time is spent on calls to the KODIAK interpreter. During the construction of a typical KIP plan, 100-200 individual KODIAK absolutes and relations are created. It takes 14 seconds to reload the knowledge base.

The MYGEN natural language generator has been modified for this implementation. MYGEN uses KODIAK to efficiently find the most specific pattern for generating a particular concept. A small number of relatively specific canned phrases have been added to MYGEN's knowledge base in order to express KIP's concerns to the user.

4.7 Conclusion

In this chapter, KIP's three plan modules have been described: (1) goal establishment, (2) plan determination, and (3) plan failure detection. KIP's plan synthesis process iterates through these three modules until a complete plan can be suggested to the user. In the following two chapters, goal establishment and plan determination are discussed. The remainder of this thesis focuses primarily on plan failure detection. In Chapters 7-10, I describe how concerns are used to detect plan failures in unique planning situations.

In this chapter, I have focused on the type of question that KIP (and UC) is asked to address: the how-to question. However, the same modules of plan synthesis can also be used to address other user problems. For example, suppose the user asks the following question:

(4) `I typed 'rm terry', but I got the message: Permission denied. What should I do?`

If KIP is faced with such a problem, it uses the same plan modules in a slightly different order. Plan failure detection can determine the most likely cause of this failure. A new goal can be established, which will be satisfaction of the particular problem in the plan. Plan determination can select and specify a modified workable version of the user's plan. Alternatively, if the user's plan cannot be modified an entirely new plan can be constructed. This thesis focuses on constructing plans for how-to questions, due to their simplicity and importance to UC.

Chapter 5

Goal Establishment

5.1 Introduction

Goal establishment refers to the process of deciding which goals should be considered in a given planning situation and which of these goals should be considered at a particular point in the planning process. KIP's general strategy in this regard is to construct and plan for *compound goals*, i.e. goals which may be dominated by a number of goals in the goal hierarchy, rather than construct a plan for a list of discrete unrelated goals. The advantage of this approach is that KIP can select known plans from its planning knowledge-base which address the compound goals, rather than have to construct a number of individual plans. The advantage of this approach over other planning strategies will be demonstrated in Chapter 6.

This process is divided into two parts:

(1) *Goal Detection* - detection of goals from the planning situation

(2) *Goal Selection* - selecting one goal from the many goals that need to be satisfied and decomposition of compound goals

Goal detection is similar to the notion described in [35]. Wilensky's work focuses on planning for multiple goals related through some goal relationship such as goal conflict. My work focuses on the detection of these goal relationships. I assume that goal relationships are only determined in terms of a particular plan which satisfies one of the goals. For example, in order to determine that two goals conflict, KIP must determine a plan for the first goal, and detect a conflict between the effects of that plan and the second goal. Goal selection has been added because of the need to plan in situations where multiple goals have been detected, and no goal relationships have yet been determined. The goal

selection process has been separated from goal detection so that KIP can first determine a plan for one goal and then determine if that plan conflicts with other detected goals.

5.2 Goal Detection

Goal detection distinguishes the three different types of goals that must be detected by KIP:

(a) *Expressed goals* - goals in the user's utterance

(b) *Inferred goals* - goals inferred during the planning process

(c) *Interest-derived goals* - goals which reflect threatened user interests

KIP's method for addressing each of these problems is discussed in terms of examples from UC. In so doing, many issues in goal establishment which are related to plan failure detection are avoided. These issues will be discussed more fully in Chapters 7-10.

5.2.1 Expressed Goal Detection

Expressed goal detection refers to the process of determining the initial goals for which KIP must plan, from the user's utterance. In the current implementation of KIP, the goals expressed by the user are passed to KIP by the PAGAN goal analyzer[25,38]. PAGAN determines these goals by examining the user's utterances. PAGAN determines what goals the user's utterances signify by using its knowledge of user plans and goals. PAGAN then returns these goals as the goals of the user.

PAGAN also determines the importance level of the various goals it has detected. This importance level information is passed to KIP by PAGAN in the representation of the expressed goal. In the current implementation of KIP, importance level is represented by a HAS-IMPORTANCE-LEVEL relation between the goal and an importance level between 0 and 1. KIP assumes that PAGAN has accurately detected the goals which have been expressed in the user's utterance, and does not evaluate them further.

For example, suppose the user asks the following question:

```
(1) How do I find out if the machine named dali
    is down?
```

PAGAN infers that the user has the goal of determining if the machine named dali is currently down; it assigns an importance level of 0.7 to this goal. PAGAN further infers that the user wants a plan that will enable him to determine the status of dali.

5.2.2 Inferred Goal Detection

In the present implementation of KIP, two types of goals are inferred during the planning process:

(a) *unsatisfied goal constituents* -	parts of a goal not satisfied by a determined plan for that goal
(b) *plan failure avoidance goals* -	goals which must be satisfied in order to avoid plan failures for a plan

5.2.2.1 Unsatisfied Goal Constituents

When KIP determines that parts of a goal have not been satisfied by a particular plan, KIP tries to infer constituents of that goal. If a goal can be divided into constituent parts, the goal constituents which have not been satisfied by the plan for the entire goal are added to the list of goals KIP must satisfy. KIP does this by considering the effects of a plan it has chosen for a particular goal. If the effects of the plan differ from the goal, KIP decomposes the original goal into its constituent parts and determines which of these parts have not been satisfied by the plan. This strategy is similar to strategies used in GPS and other knowledge-deficient planners.

In example (1), suppose that KIP had chosen to use the USE-RUPTIME-COMMAND plan during the plan determination phase. This plan is actually designed to determine the uptime or downtime of all the machines on the network of the current machine. Since this is not a stored plan for the user's goal, KIP must determine if the USE-RUPTIME-COMMAND plan needs modification in order to properly address the user's goal.

During a subsequent goal detection phase, KIP must determine what goals of the user have not been satisfied by the selected plan. In example (1), KIP determines that the user's goal of determining the downtime of a particular machine has not been satisfied. This determination is made by matching the effect of the individual USE-RUPTIME-COMMAND1 plan, in the particular planning situation, with the expressed goal it was meant to satisfy. In this case, the USE-RUPTIME-COMMAND1 plan operates on the set of machines to which the current machine is connected, while the user's expressed goal refers to particular machine named dali. This is a common difference in UC, since many information-type commands return a list of objects, while users are often only interested in information about a particular object. (This difference is outlined in Figure 5.1.) Therefore, KIP infers the goals of filtering the information provided by the USE-RUPTIME-COMMAND1 about the set of machines to which the current machine is connected. KIP does this so that only information about the machine named dali is presented to the user. A KIP trace of goal decomposition is presented later in this chapter in the context of goal selection.

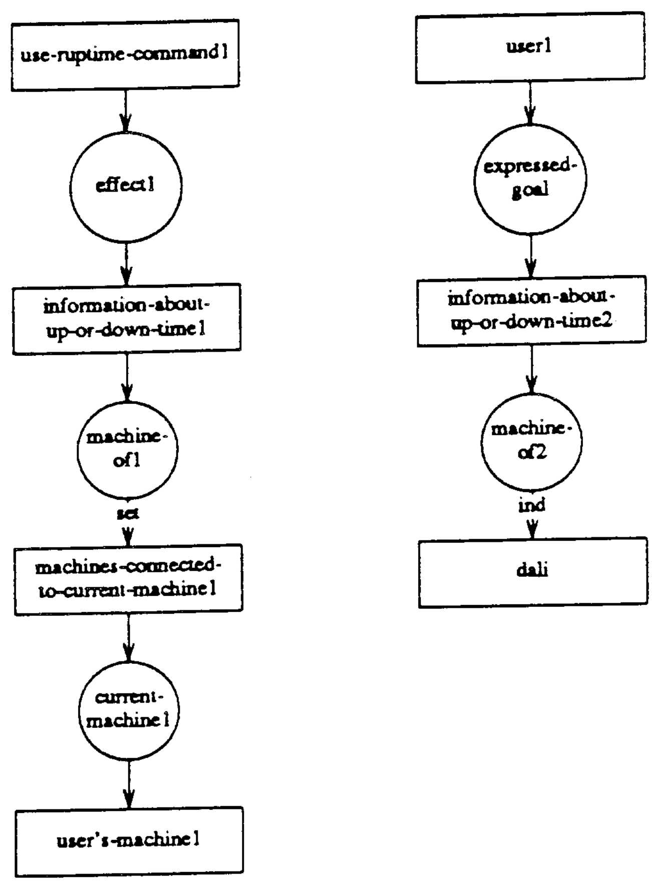

Figure 5.1: Effect of USE-RUPTIME-COMMAND1 and the User's Goal

5.2.2.2 Plan Failure Avoidance Goals

Plan failure avoidance goals are goals which KIP must satisfy in order to avoid plan failures for a determined plan. These goals are inferred during the plan failure detection phase. Unlike other planners which check all the conditions of a plan in order to detect plan failure, KIP only checks those conditions which cause a concern in a particular planning situation. Plan failure avoidance goals reflect KIP's concerns about the determined plan. The two types of plan failure avoidance goals correspond to the two types of plan failure:

(1) *condition failure avoidance goals* -	goals that reflect condition concerns
(2) *goal conflict failure avoidance goals* -	goals that reflect goal conflict concerns

If KIP has a condition concern about the satisfaction of a condition of a particular plan, KIP instantiates a condition failure avoidance goal that reflects this concern. If KIP has a goal conflict concern about the effects of a particular plan, KIP instantiates a goal conflict failure avoidance goal regarding the resolution of the goal conflict reflected in this concern. Plan failure avoidance goals will be described more fully in the discussion of plan failure detection in Chapters 7 and 8.

5.2.3 Interest-Derived Goal Detection

Interest-derived goal detection refers to the process of instantiating goals that reflect long-term user interests. *Interests* are general states that KIP assumes are important to the user. When interests become active, they can give rise to one or more goals when they become active. For example, when an interest is threatened, it gives rise to goals that will prevent that interest from being threatened. An important difference between goals and interests is that plans are meant to satisfy goals but plans cannot satisfy interests. In a sense, interests are never satisfied; they are just no longer considered once the goals to which they give rise are satisfied. For example, preserving one's files is an important user interest in the context of UC. Suppose that the file-preservation interest is threatened in a particular situation by the deletion of the file file1. The threat to the file-preservation interest gives rise to the goal of preserving the contents of the file named file1. Once this goal is satisfied, there is no threat to the file-preservation interest. Therefore, the file-preservation interest is no longer considered in this particular planning situation. However, it is still an interest of the user. An interest might be considered again if it was threatened by another action. On the other hand, once a particular goal is satisfied, it is no longer considered at all.

5.2.3.1 Interests and Themes

Schank and Abelson[33] introduced the term *theme* which refers to background knowledge about goals that are likely to arise in general situations. For example, the SHERIFF-ROLE-THEME might refer to the goals of protecting citizens, arresting criminals, and round up posses. Schank and Abelson used themes to anticipate goals of character in simple stories. Wilensky[35] operationalizes the theme notion for planning by using situation-goal pairs. If a planner notices that it is in a particular situation, certain goals are immediately added to a planner's list of goals.

In KIP, themes have been extended so as to indicate which interests should be considered. When KIP notices that it is in a particular situation, it activates a particular theme. This theme then indicates which interests should be considered. If these interests are important in the particular planning situation, goals are instantiated which reflect these interests. KIP is mainly concerned with detecting interests which have threatened by the effects of a particular plan. Therefore, if KIP must decide if a particular plan threatens a user interest, KIP must only consider those interests which are referred to in the current active themes.

However, in the current implementation of KIP, due to its narrow UNIX domain, all interests are referred to by the UNIX-PLANNER-THEME. Examples of interests organized under this theme include the file-preservation, preserve-system-resources, and conserve-disk-space interests. The UNIX-PLANNER-THEME is always active in UC. In Chapters 9 and 10, I present examples of goal conflicts where these interests are threatened.

Interests not only give rise to goals when they are threatened, but also when they are enabled by a state of the world. For example, a planner interested in amassing wealth might instantiate the goal of taking-the-money upon seeing an open safe. These interests have not been implemented due to the paucity of such interests in the UNIX domain.

5.2.3.2 Interest Selection and Evaluation

Let us consider a simple example where interest-derived goal detection is important. Suppose the user asks the following question:

(2) `How do I move the file david to the file jim?`

PAGAN would detect the single goal of moving the file david to the file named jim. However, since KIP knows that the user has an interest in keeping the contents of all of his files, there is a good chance that he wants to have access to the file jim. As is described below, KIP may create the goal of preserving access to the file jim in order to reflect this interest on the part of the user. Once this goal is created, any plan that conflicts with it will cause a goal conflict. KIP must then resolve the conflict between these two user goals.

There are two parts of the interest-derived goal detection process:

(1) *interest selection* - selection of those interests which might be important to the user in a particular planning situation

(2) *interest evaluation* - evaluation of selected interests in the particular planning situation

Interest evaluation is particularly important since the goal should be created only if the interest is appropriate for the particular planning situation. For example, the individual goal of having access to file jim should be created only if KIP has reason to believe that a file named jim already exists. Therefore, if KIP has selected the file preservation interest, and KIP knows that the file jim does not exist, the goal of preserving the file jim should not be inferred.

Interest evaluation is often a complex process. Even in the relatively simple file preservation example, even if KIP knows the file jim exists, KIP might also need to consider whether the user wants to preserve the file jim. Ideally, KIP should try to evaluate as few interests as possible; KIP should select only those interests which are likely to be threatened in the particular planning situation.

However, it is difficult to select those interests which are likely to be threatened in a given planning situation from among the large number of potential user interests. KIP's knowledge-base includes many interests that KIP assumes on the part of the user. Consideration of every potential interest would be computationally inefficient and contrary to KIP's knowledge-intensive approach. Therefore, KIP exploits a knowledge-intensive means of limiting those interests considered.

One possible knowledge intensive method for detecting threatened interests is to store relationships between situations and threatened interests. According to the situation/interest proposal, every time a planner was faced with a particular situation, it adds the associated interests to the set of interests under consideration. KIP then must evaluate which interests are potentially threatened. For example, if the user expressed a particular goal, the planner would match that goal against interests that goal might threaten in the current set of considered interests.

In KIP, I take an alternative position. Interests are considered only when they are threatened by some action. Therefore, KIP only considers user interests when they have been threatened by an effect of a user plan. These include plans that KIP has determined for the user, and plans the user has determined himself. For example, in example (2), KIP determines that the interest of preserving the file named jim is threatened. However, the file-preservation interest is only considered after KIP has determined a plan for the user's goal, the USER-MV-COMMAND1 plan. The plan/interest proposal is implemented using goal

conflict concerns. I will discuss interest selection and interest evaluation using goal conflict concerns in Chapter 10.

The situation/interest proposal, and KIP's plan/interest proposal, may be different ways of expressing the same knowledge. I have chosen the plan/interest proposal due to KIP's focus on plan determination. A robot planner that generates its own goals might also need to detect interests which are threatened as a result of the goals it has generated.

5.2.4 Concretion in Goal Detection

In order to find the most specific plan for a particular goal, KIP must first confirm that all of the goals for which it is trying to plan are as specific as possible. The goal should be made a specific as possible through concretion inferences. Concretion inferences[30] refer to inferences that an individual is dominated by one of the subtypes of its parent. For example, if an individual building is described, and it contains three apartments, the building can be concreted to be an instance of apartment building.

Each goal should be concreted during goal detection so that plan selection can consider the most specific plans possible. Actually, much of the concretion processing on goals is done before KIP actually receives the goals as input. In order to detect expressed goals, the parser and goal analyzer perform concretion inferencing so as to best understand the goals of the user. Inferred goals and interest-derived goals are concreted by KIP when these goals are instantiated.

The concretion of potential goals is not merely an artifact of KIP's or UC's architecture but is an important principle for the design of the algorithm for any commonsense planner. Concreting unsatisfied goals is an operationalization of Wilensky's First Law of Knowledge Application[35]: *Always apply the most specific pieces of knowledge available*. In order for a planner to apply the most specific pieces of knowledge in a particular situation, the algorithm must first make the situation understood in the most specific way possible. In order for a planner to select the most specific plan for a particular goal, it must first make that goal as concrete as possible.

In the current implementation of KIP, KIP uses a concretion mechanism which I implemented as part of a new implementation of the KODIAK knowledge representation interpreter. The concretion mechanism looks at all the relations of a particular concept (i.e. the relations for which the concept is the domain). For each relation, the mechanism decides if the relation can be an instance of a more specific relation according to the domain of the relation. If the concretion mechanism decides to make a relation an instance of a more specific relation, then the mechanism asserts that range of the relation is also more specific. For example, suppose that OBJECT-1 has a Name-1 relation whose range is NAME-STRING-1. If KIP learns that OBJECT-1 is a FILE, then the concretion mechanism decides that the Name-1 relation can be an instance of the File-Name relation. Furthermore, the concretion mechanism asserts that NAME-STRING-1 is an instance of FILE-NAME-STRING.

5.3 Goal Selection

Goal Selection is the process of determining which of the goals considered by KIP should be addressed by the plan determination component at a particular phase of the planning process. For example, if KIP has been passed 5 goals by the goal analyzer, and there is no plan to solve all these goals simultaneously, KIP must select one or more of these goals for which to plan. Later, once a plan has been determined for the particular goal, satisfaction of other goals can be added to the plan.

In addition, goal selection is sometimes necessary when KIP has detected one compound goal. If KIP cannot find a plan for the entire goal, KIP must divide the compound goal into its component goals. KIP must decide which of these component goals should be addressed before the others.

There are two main criteria for goal selection:

(1) *importance level* - how important the goal is to the user

(2) *ease of satisfaction* - how easily the goal can be satisfied

5.3.1 Importance Level as a Criteria for Goal Selection

The determination of importance level differs according to three classes of goals which are candidates for goal selection:

5.3.1.1 Expressed Goal Selection

For goals that are passed to KIP by the goal analyzer, importance level is determined by the goal analyzer and declared in the representation of the goal. This information is either inferred from the user's utterance or from the goal analyzer's knowledge of plans and goals.

For example, suppose the user asks the following question:

(3) `How do I send a formatted version of the file letter.ms named letter.out to the printer?`

Suppose that PAGAN determines that the user wants first to send the formatted version of the file to the printer, and then to delete the formatted version. PAGAN might pass the PRINT-FILE and DELETE-FILE goals as separate goals. PAGAN determines that the PRINT-FILE goal is more important by examining the user's utterance. This importance level information is passed to KIP by PAGAN in the representation of the two goals. KIP selects a single goal by referring to their importance level in the particular planning situation.

Therefore, KIP selects the PRINT-FILE goal. During a subsequent iteration, the DELETE-FILE goal is addressed. This process is demonstrated in a KIP trace which highlights goal establishment in Figure 5.2 below.

Figure 5.2: KIP Trace of Goal Selection with Multiple Goals

User: How do I send a formatted version of the file letter.ms named letter.out to the printer?

PAGAN produces:
(print-file-effect-1
(File-Arg-Of-Print-File-Effect-25
(file-1 (File-Name-100 letter.out-1)))
(More-Important-Than-80 delete-file-effect-1))

;; *This is the first of the two goals that PAGAN has detected.*
;; *PAGAN has specified that the file to be printed is file-1.*
;; *file-1's name is letter.out-1.*
;; *Furthermore, PAGAN has specified that the print-file-effect-1 is more important*
;; *than delete-file-effect-1.*
;; *The more-important-than relation is defined in terms of the relative*
;; *HAS-IMPORTANCE-LEVEL of its domain and range.*

(delete-file-effect-1
(Effected-File-55 file-1))

;; file-1 *is also the file that is to be deleted.*
;; *In this case, this is called the effected-file, because unlike*
;; *the print-file-effect, which does not affect the file printed, the delete-file-effect*
;; *effects the file by deleting it.*

(kip (delete-file-effect-1 print-file-effect-1))

;; KIP *is being called with the input of the two goals*

Entering Goal Establishment Phase:

KIP is trying to determine a plan for the list of goals:
(delete-file-effect-1 print-file-effect-1)

Selecting a goal from the List Of Goals ((delete-file-effect-1 print-file-effect-1))

;; *The List Of Goals is the current set of goals, for which KIP*
;; *is determining a plan*

Sorting the goals according to importance level
The New List Of Goals is now (print-file-effect-1 delete-file-effect-1)

;; *Because KIP knows that the print-file-effect-1 is more important than*
;; *delete-file-effect-1, KIP reorders the list of goals to reflect*
;; *this knowledge*

The Most Important Goal is now print-file-effect-1
;; *It now becomes the current goal*

Entering Plan Determination Phase:

Looking for a plan for the Current Goal (print-file-effect-1)

```
First looking at stored plans
Selected (lpr-command) as a potential plan
Now specifying the plan for the particular planning situation:
(lpr-command-85
 (Destination-Of-Lpr-Command-88 printer-87)
 (Format-Of-Lpr-Command-94
  (unix-file-printing-command-format-93
   (Command-Arg-104 lpr-string-101)
   (Format-File-Arg-98 letter.out-1)))
;; The user could execute this plan by typing lpr letter.out
 (File-Arg-Of-Unix-File-Command-96 file-1)
;; the file to be manipulated by the lpr-command is the same
;; as the file of the print-file-effect goal
 (Intended-Effect-Of-Lpr-Command-86
  (print-file-effect-1
   (Destination-Of-Print-File-Effect-108 printer-87)
   (File-Arg-Of-Print-File-Effect-25
    (file-1
     (Contents-29
      (file-contents-28
       (Printing-Of-31 printing-26)))
;; The contents of file-1 have some associated printing-26 which is the
;; KODIAK representation of the object which is printed
     (File-Name-100 letter.out-1)))
   (Final-State-Of-Print-File-Effect-39
    (printed-on-state-32
     (Paper-Of-Printed-On-36 paper-33)
     (Printing-Of-Printed-On-38 printing-26)))
   ;; The final state of this effect is that the printing is now on paper-33
   (Initial-State-Of-Print-File-Effect-51
    (printed-on-state-44
     (Paper-Of-Printed-On-48 paper-33)
     (Printing-Of-Printed-On-50 blank-46)))
   ;;  The initial state of print-file-effect-1 is that the paper is blank
   (More-Important-Than-80 delete-file-effect-1))))
Entering Plan Failure Detection Phase:
No plan failures detected.
Entering Goal Establishment Phase:
Selecting a goal from the List Of Goals ((delete-file-effect-1))
selecting the remaining goal:
delete-file-effect-1
;; KIP has selected the second of the two detected goals, the
;; goal of delete the printed file
Entering Plan Determination Phase:
```

Looking for a plan for the Current Goal delete-file-effect-1
First looking at stored plans
Selected (rm-command) as a potential plan
Now specifying the plan for the particular planning situation:
(rm-command-113
 (Format-Of-Unix-File-Command-118
 (unix-file-command-format-117
  (Format-File-Arg-124 letter.out-1)
  (Command-Arg-122 rm-string-119)))
 ;; The user could execute this plan by typing rm letter.out
 (File-Arg-Of-Unix-File-Command-116 file-1)
 (Intended-Effect-Of-Rm-Command-114
 (delete-file-effect-1
  (Effected-File-55 file-1)
  (Final-State-Of-Delete-File-Effect-63
  (file-exist-state-56
   (File-Of-File-Exist-60 file-1)
   (Exist-Of-File-Exist-62 false-68)))
  ;; The final state of delete-file-effect-1 is that the file-1 *does not exist*
  (Initial-State-Of-Delete-File-Effect-76
  (file-exist-state-69
   (Exist-Of-File-Exist-75 true-79)
   (File-Of-File-Exist-73 file-1))))))
  ;; The initial state is that the file does exist
Entering Plan Failure Detection Phase:
No plan failures have been detected
**To print and delete letter.out, first type lpr letter.out,
then type rm letter.out.**

5.3.1.2 Inferred Goal Selection

A goal constituent's importance is determined according to its importance to the compound goal. This importance is represented in the description of the decomposition of the compound goal. For example, suppose that in example (3), PAGAN does not return two separate goals. Instead suppose that one goal is returned: the PRINT-AND-DELETE-FILE goal. In many operating systems, the PRINT-AND-DELETE-FILE goal is the default effect of the command which send files to the printer. However, in UNIX there is no stored plan for the PRINT-AND-DELETE-FILE goal. Therefore, during goal selection, KIP decomposes the PRINT-AND-DELETE-FILE goal into the two component goals: the PRINT-FILE and DELETE-FILE goals. (The decomposition of the PRINT-AND-DELETE-FILE goal is described in Figure 5.3.) The PRINT-FILE is stored as more important to the decomposition of the PRINT-AND-DELETE-FILE goal than the DELETE-FILE goal. This is done by storing a more-important-than

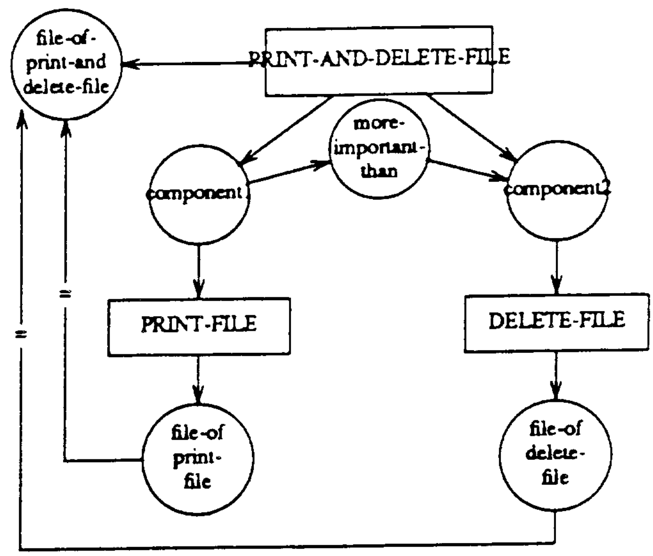

Figure 5.3: Decomposition of PRINT-AND-DELETE-FILE

relation between the PRINT-FILE component and the DELETE-FILE component. (The more-important-than relation is defined in terms of the relative HAS-IMPORTANCE-LEVEL of its domain and range.) Therefore, the PRINT-FILE goal is selected first during goal selection. This process is demonstrated in a KIP trace which highlights goal decomposition in Figure 5.4.

Figure 5.4: KIP Trace of Goal Decomposition

User: How do I send a formatted version of the file letter.ms named letter.out to the printer?

PAGAN produces:

```
(print-and-delete-file-1
 (File-Of-Print-And-Delete-File-25
  (file-1
   (File-Name-132 letter.out-1))))
```

;; PAGAN has produced one complex goal, the print-and-delete-1 goal.
;; PAGAN has also specified that the file of this effect is file-1.
;; Unlike Figure 5.2 on page 68, KIP has only been passed one goal.
;; This trace highlights what KIP does differently because it must decompose this
;; complex goal. Therefore, the representation of goals that have been described in
;; the Figure 5.2 will be abbreviated.

Entering Goal Establishment Phase:

KIP is trying to determine a plan for the list of goals:
(print-and-delete-file-1)

;; Notice that KIP is only passed one goal, it decomposes this goal later

Selecting a goal from the List Of Goals ((print-and-delete-file-1))
selecting the remaining goal
print-and-delete-file-1

Entering Plan Determination Phase:

Looking for a plan for the Current Goal print-and-delete-file-1
First looking at stored plans

;; KIP has not found a stored plan for the complex goal. Since KIP knows that
;; this goal can be decomposed into simpler goals, KIP next decomposes the
;; goal into two goals. KIP also fully specifies each of the two goals.

Entering Goal Establishment Phase:

Decomposing the Current Goal (print-and-delete-file-1) into
delete-file-effect-26 and print-file-effect-58

;; The representations of these goals are the same as in the previous example
;; where these goals were passed as separate goals.

Selecting a goal from the List Of Goals ((delete-file-effect-26 print-file-effect-34))
Sorting the goals according to importance level
The New List Of Goals is now (print-file-effect-34 delete-file-effect-26)
The Most Important Goal is now print-file-effect-34

Entering Plan Determination Phase:

Looking for a plan for the Current Goal (print-file-effect-34)
First looking at stored plans
Selected (lpr-command) as a potential plan
Now specifying the plan for the particular planning situation:
lpr-command-99

Entering Plan Failure Detection Phase:

No plan failures have been detected

Entering Goal Establishment Phase:

Selecting a goal from the List Of Goals (delete-file-effect-26)
selecting the remaining goal
delete-file-effect-26

Entering Plan Determination Phase:

Looking for a plan for the Current Goal delete-file-effect-26
First looking at stored plans
Selected (rm-command) as a potential plan
Now specifying the plan for the particular planning situation:
rm-command-127

Asserting that lpr-command-99 comes before rm-command-127

;; Part of the description of the decomposition of print-and-delete-file is that
;; the printing comes before the deleting. If KIP was not provided ordering

;; information in the goal decompostion, KIP would have to determine the plan
;; step ordering by examining preconditions.

Entering Plan Failure Detection Phase:

No plan failures have been detected

To print and delete letter.out, first type lpr letter.out,
then type rm letter.out.

;; KIP has returned the same plan as in the previous trace, even
;; though it was given two goals in that example, and one
;; complex goal in this example.

A plan failure avoidance goal's importance is determined by the degree of concern regarding the determined plan reflected by the goal reflects and the importance level of the goal that the plan was designed to satisfy. Thus, if KIP has a high degree of concern about a particular plan, and this plan was designed for a goal with a high importance level, the plan failure avoidance goal will also have a high importance level. The determination of degree of concern in a particular planning situation will be described more fully in Chapter 8.

5.3.1.3 Interest-Derived Goal Selection

The importance level of a goal inferred to reflect a user interest is determined by the importance level of the interest and its evaluation in the particular problem situation. (The representation of the importance level of interests is the same as the representation of the importance level of goals described earlier.) The importance level of an interest-related goal is compared to the importance level of other goals being considered. KIP's approach to the determination of the importance level of interest-related goals is described in Chapter 10.

5.3.2 Ease of Satisfaction as a Criteria for Goal Selection

If all the goals under consideration are of the same importance level, the ease of goal satisfaction is used to determine which goal should be selected first for planning. This strategy corresponds to the commonsense notion of trying to first solve the easiest parts of a problem. In order to determine the ease of satisfaction, a planner should have knowledge that certain goals can be satisfied more easily than other goals. For example, if a planner works in a number of different domains, the planner might know that goals from one domain are generally easier to satisfy than goals from other domains. In the current implementation of KIP, the only ease of satisfaction criteria is whether there is a stored plan for a particular goal. This corresponds to the commonsense notion of trying to solve those parts of a problem which one already has a good idea of how to solve. Accordingly, if one

of the possible goals has a stored plan, and all the goals have the same importance level, that goal is selected before other goals.

In the current implementation of KIP, a simple ease of satisfaction criterion for goal selection is used. If one of the individuals goals has a direct parent for which there is a stored plan, this goal is selected. Direct parent is defined as a parent which is directly above the individual goal in the KODIAK hierarchy.

5.4 Conclusion

In this chapter, the two parts of goal establishment have been discussed: (1) goal detection, and (2) goal selection. The advantage of this approach will be demonstrated in the following chapter on plan determination. Due to KIP's ability to detect and instantiate complex goals, KIP can both use stored plans for these complex goals, or use other previously-known plans which were designed for similar goals. If this strategy is unsuccessful, KIP can decompose the complex goal into its component subgoals during goal selection. KIP can then attempt to determine plans for these subgoals.

Chapter 6

Plan Determination

6.1 Introduction

Plan determination refers to the process of determining a plan for the goal of the user. During each iteration of the planning process, a new plan is determined for the selected goal. There are two steps to plan determination:

(1) *plan selection* -	selecting a general plan among the plans KIP knows for a selected goal
(2) *plan specification* -	specifying the values of the general plan for the particular problem situation

During plan selection, KIP must select a plan for the selected goal from among the many plans of which KIP is aware. Unlike earlier planners which built plans from a number of simple actions, KIP tries to modify a small number of complex plans in order to satisfy the current goal of the user. Since many plans are stored in KIP's database, finding the best previously-known plan that will satisfy the current goals of the user is a non-trivial problem.

Unlike earlier planners, KIP's strategy for plan selection is not to search among the known plans. Instead, KIP exploits the hierarchy of goals to find a stored plan associated with a goal. Since a plan is a proposed action in service of a goal, KIP stores all known plans as plans for particular goals. Therefore, instead of evaluating many plans, KIP looks for plans associated with its detected goals.

In this chapter, I first describe KIP's strategy for plan selection when KIP has a stored plan for the current goals. I then describe KIP's plan selection strategy when it is faced with a unique situation for which no stored plan is available. KIP's plan specification strategy is the same in both these situations. Therefore, I describe plan specification in the following section regarding stored plans.

6.2 Using Stored Plans to Satisfy User Goals

In order to select a plan for a selected goal, KIP first examines the goal itself and determines if there is a known plan for that goal. In order to find a known plan for the selected goal, KIP must first find the parent of the individual goal. In most cases, KIP has a stored plan for a parent of the selected goal instance. This parent goal organizes information about the particular goal, without specifying certain values of the goal. Examples of such specifics include the name of the person who has the goal, and the time of day that the person has the goal. In UC, such specifics as filenames and times are generally not included in the parent goal. However, for certain goals, some of these specifics may be specified. For example, KIP has particular information about editing the /usr/lib/aliases file, a file on most UNIX machines which contains forwarding addresses for UNIX computer mail. Therefore, KIP stores this specific filename in the representation of the goal of changing the forwarding address database.

Thus, stored plans are used in order to focus immediately on a known plan which has been designed for a particular goal. KIP's knowledge-base includes stored plans for many complex goals. Plans for these goals could be constructed by determining plans for the individual components of the complex goals. However, by storing detailed complex plans for these complex goals, KIP avoids a complicated plan synthesis. More importantly, KIP can use the specific knowledge of the planner knowledge-base designer - namely, that a particular set of actions is the best plan for a complex goal. In this way, KIP can capture the expert's expertise regarding the most efficient and trouble-free plan for a particular complex goal. KIP can do this without encoding and evaluating all the knowledge that resulted in the UNIX expert's choice of the particular complex plan.

For example, suppose the user asks the following question:

(1) `How do I find out which processes are using the most CPU time on my machine?`

A reasonable response might be:

```
Type ps uaxg | fgrep -v '0.'  .  This will
produce a list of all processes using more
than 1% of the CPU.
```

In example (1), KIP is asked to find a plan for a relatively complex goal, the FIND-BIG-PROCESSES1 goal. In this case, KIP's knowledge base includes a plan for the parent of such a goal, FIND-BIG-PROCESSES goal. Even though the FIND-BIG-PROCESSES1 goal is a relatively complex goal, KIP's planning work is minimal since it already knows a debugged complex plan that will work for this goal.

Stored plans are found by looking at the PLANFOR relation between plans and goals in the KODIAK hierarchy. A PLANFOR relates a goal to the plan for that goal. Thus, if KIP is passed an individual goal far down in the hierarchy, this goal's parents may indicate a specific plan through a PLANFOR relationship. I show examples of such PLANFOR relationships in the following section.

6.2.1 Plan Specification of Stored Plans

Once KIP has selected a potential plan, it must specify the values of certain parts of the plan. KIP has selected a plan based on its function as the stored plan for some goal. As described earlier, in every non-individual goal, many of the values are not specified. Therefore, a plan stored for a non-individual goal will also have many unspecified values. During plan specification, KIP uses information about the specific values of a particular goal to fill in the specific values for the plan it has selected. This information is stored in *equate* associations between the unspecified values in the plan and the unspecified values in the goal. Equate links are described in Section 3.5. In Figure 6.1, I give an example of a plan specification of the MV-COMMAND plan. The MV-COMMAND is designed to move the source file to the destination file using the format mv source-file destination-file. This representation of the MV-COMMAND is complex due to the fact that the command arguments, i.e. that names of the files that are moved, change during the execution of the plan. File-names used in the format of the MV-COMMAND refer to File-Name relations before the plan is executed. Figure 6.2 shows part of the representation of the MV-COMMAND stored in the KODIAK knowledge base. Figure 6.3 shows a graphical representation of the individaul MV-COMMAND created in Figure 6.1.

Figure 6.1: KIP Trace of Stored Plan Specification

How do I rename the file named lewis to be called bernstein?

```
;; KIP receives the following rename-file-effect goal as input
(rename-file-effect-1
  (New-Name-37 bernstein-1)
  (Previous-Name-63 lewis-1)
  (Destination-File-Of-Rename-File-Effect-48 file-2)
  ;; The destination file is the file which has the new-name before the
  ;; state-change occurs
  (Destination-File-File-Name-46
   (file-name-state-38
    (Value-Of-File-Name-50 bernstein-1)
    (Object-Of-File-Name-42
     (file-2
      (File-Name-41 bernstein-1)
```

;; file-name-state-38 *refers to the the fact that the destination file has the*
;; *name* bernstein-1. *This state is true before the state-change interval*
(File-Exist-70 false-69)))
;; *KIP has been told that the destination file does not exist. Therefore there*
;; *will be not concerns about deletion of the destination file*
(Source-File-Of-Rename-File-Effect-24
(file-1
(File-Name-56 lewis-1)
(File-Name-28 bernstein-1))
;; *The source file is the file that is effected in this state change. Its file-name*
;; *is first lewis and later bernstein*
(Initial-State-Of-Rename-File-Effect-61
(file-name-state-53
(Value-Of-File-Name-65 lewis-1)
(Object-Of-File-Name-57 file-1)))
(Final-State-Of-Rename-File-Effect-33
(file-name-state-25
(Value-Of-File-Name-52 bernstein-1)
(Object-Of-File-Name-29 file-1)))
;; *The initial state and final state's objects are both file-1. The values are*
;;*the values of* Previous-Name-63 *and* New-Name-37 *respectively.*
(State-Change-Interval-Of-Rename-File-Effect-35 state-change-time-34))
;; *The state-change-interval is particularly important in this example,*
;; *because the names used by in the command format, must be the names of*
;; *the files before the state-change-interval*

(KIP (LIST RENAME-FILE-EFFECT-1))

Entering Goal Establishment Phase:

KIP is trying to determine a plan for the list of goals:
(rename-file-effect-1)

Selecting a goal from the List Of Goals ((rename-file-effect-1))
selecting the remaining goal
rename-file-effect-1

Entering Plan Determination Phase:
Looking for a plan for the Current Goal (rename-file-effect-1)

First looking at stored plans
Selected mv-command as a potential plan
Now specifying the plan for the particular planning situation:
(mv-command-71
(First-File-Arg-File-Name-100
(file-name-state-92
(Value-Of-File-Name-98 lewis-1)
(Object-Of-File-Name-96 file-1)))
(Second-File-Arg-File-Name-83
(file-name-state-75
(Value-Of-File-Name-81 bernstein-1)

```
  (Object-Of-File-Name-79 file-2)))
;; These two states represent the file-name of the command arguments.
;; These states must be true before the execution of the plan
 (First-File-Arg-Of-Mv-Command-91 file-1)
 (Second-File-Arg-Of-Mv-Command-74 file-2)
 (Format-Of-Unix-Two-File-Command-89
  (unix-two-file-command-format-88
   (Command-Arg-108 mv-string-105)
   (Format-First-File-Arg-102 lewis-1)
   (Format-Second-File-Arg-110 bernstein-1)))
 ;; The user could execute this command by typing mv lewis bernstein
 (Command-Name-Of-Mv-Command-106 mv-string-105)
 (Intended-Effect-Of-Mv-Command-72 rename-file-effect-1)
 ;; rename-file-effect-1 is represented above
 (Command-Interval-Of-Mv-Command-85 plan-time-84)
 (Intended-Effect-Interval-87 state-change-time-34)
 ;; The command-interval starts before the intended-effect-interval
Entering Plan Failure Detection Phase:
No plan failures are detected
To move the file named lewis to the file named bernstein,
use mv lewis bernstein.
```

6.2.2 Comparison with Previous Approaches which use Complex Plans

KIP's strategy of using complex plans for complex goals is similar to strategies used by earlier planners such as HACKER[34] and MACROPS [11]. These planners created subroutines or macro-operators for complex goals which they had encountered previously. Recently, many researchers have focused on methods for creating such complex plans. Explanation-based learning methods (EBL) [9] focus on creating complex plans in planning by generalizing over example problems which are presented to the system. Explanation-based generalization (EBG) research [28] has focused on generalizing over a set of axioms in a particular domain.

One problem that been not been addressed by any of these systems is how these complex plans are actually used once they have been created. Adding many new plans to the knowledge base in all of these systems would cause an increase in the processing necessary for plan selection. Instead of trying every plan and seeing which has best result, some means of indexing the plans would be necessary. In KIP, indexing of plans is not important for the selection of stored plans, since plans are chosen by their relationship to the selected goal.

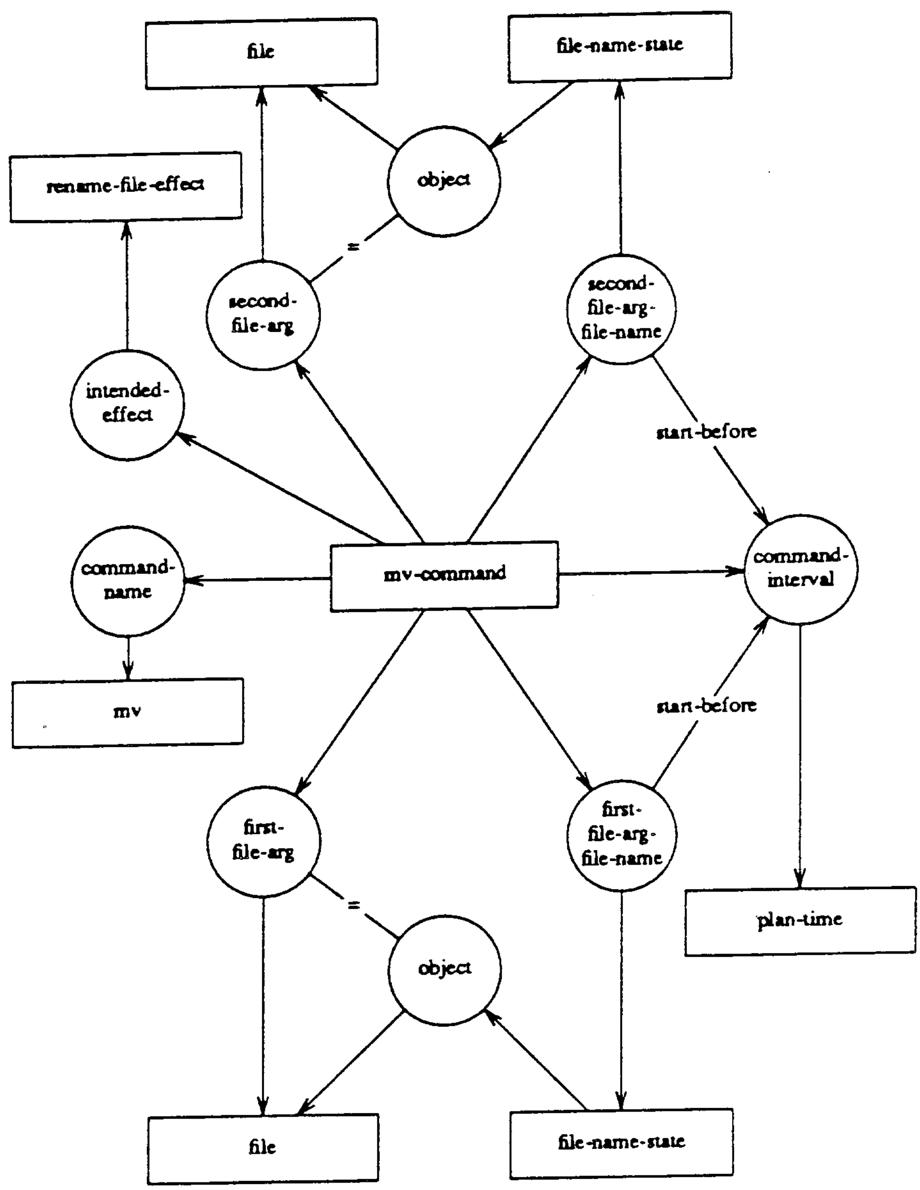

Figure 6.2: Representation of MV-COMMAND

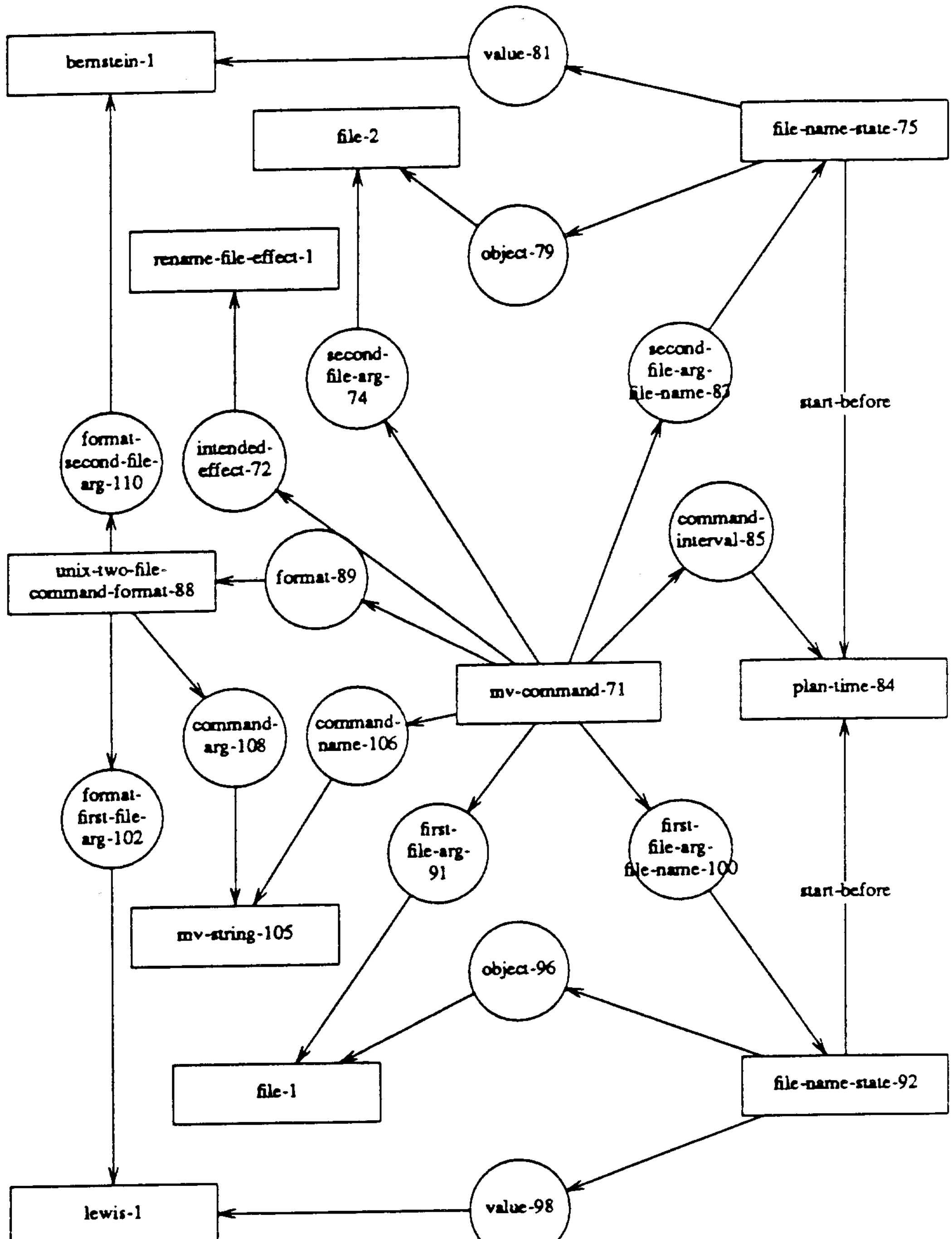

Figure 6.3: Representation of Individual MV-COMMAND

6.2.3 Storing Multiple Plans for a Goal

Sometimes, KIP stores more than one plan for a particular goal. This may occur when the planner knowledge-base designer knows of more than one plan that will satisfy a particular goal. Multiple plans are often stored for general goals, since most specific goals tend to have one specific plan associated with them.

In addition, KIP can also store the knowledge that, in general, one plan is better to use than another plan. The ordering of stored plans for a particular goal encodes the knowledge that one plan is, in general, preferable over other stored plans. In the current implementation of KIP, the ordering of multiple plans is implemented by storing multiple planfor relations in a list. The best plan is stored at the beginning of the list. As mentioned in Chapter 3, there is currently no way to specify a list of values as the range of a relation. Ordering of multiple plans is implemented using the current KODIAK interpreter, and defining the PLANFOR relations in such a way that the first PLANFOR relation returned is the most important one.

This ordering knowledge enables KIP to attempt to use these plans in an order prescribed by the planner knowledge-base designer. The knowledge base designer does not need to encode the reasons that one plan is better than another.

For example, suppose the user asks the following question:

(2) `How do I edit a file named mayfield?`

KIP knows of a number of different plans for the goal of editing a file. KIP stores these plans in the following order: the VI-COMMAND plan, the EMACS-COMMAND plan, and the ED-COMMAND plan. Thus, KIP first chooses the VI-COMMAND plan. This plan is chosen because KIP's knowledge base stores it as the best plan for this particular goal. The knowledge base designer has stored VI-COMMAND plan as the best plan because it both requires a small amount of startup time and the editor is familiar to most UNIX users. (In this case, these reasons are not stored in the KIP knowledge base.) However, if KIP detects problems with this plan during plan failure detection, another stored plan is considered. A KIP trace of Example (2) is presented in Figure 6.4.

Figure 6.4: KIP Trace of Plan Determination with Multiple Plans for a Single Goal

User: How do I edit the file named mayfield?

KIP is trying to determine a plan for the list of goals:

```
(edit-file-effect-1
 (Effected-File-23
 (file-1
  (File-Name-47 mayfield-3)))
 (Final-State-Of-Edit-File-Effect-31
```

```
 (contents-state-24
  (File-Of-Contents-28 file-1)
  (Contents-Of-Contents-30 file-contents-26)))
(Initial-State-Of-Edit-File-Effect-43
 (contents-state-36
  (File-Of-Contents-40 file-1)
  (Contents-Of-Contents-42 file-contents-38))))
```

;; *The initial state is that* file-1 *has some contents, and the final-state is that* file-1
;; *has some other contents. These contents are not necessarily different.*

Selecting a goal from the List Of Goals ((edit-file-effect-1))
selecting the remaining goal
edit-file-effect-1

Looking for a plan for the Current Goal (edit-file-effect-1)

First looking at stored plans
There are three potential plans for the current goal

The plans are:
(vi-command emacs-command ed-command)

Will try these plans in order
Trying vi-command first

;; *KIP has used the knowledge that the vi-command is generally the*
;; *preferable plan. If KIP finds that this plan fails during plan failure detection, it*
;; *will attempt to use one of the other plans.*

Now specifying the plan for the particular planning situation:

```
(vi-command-60
 (Intended-Effect-Of-Unix-Editing-Command-62
  (edit-file-effect-1
   (Effected-File-27
   (file-1
    (File-Name-76 mayfield-3)))
   (Final-State-Of-Edit-File-Effect-37
    (contents-state-30
     (File-Of-Contents-34 file-1)
     (Contents-Of-Contents-36 file-contents-32)))
   (Initial-State-Of-Edit-File-Effect-53
    (contents-state-44
     (File-Of-Contents-50 file-1)
     (Contents-Of-Contents-52 file-contents-46)))))
 (File-Arg-Of-Unix-File-Command-64 file-1)
```

;; *The file-arg is the same as the effected-file of* edit-file-effect-1

```
 (Format-Of-Unix-File-Command-66
  (unix-file-command-format-65
   (Command-Arg-72 vi-string-67)
   (Format-File-Arg-74 mayfield-3))))
```

;; *The format is vi followed by mayfield. This information is inherited from the*
;; *unix-file-command seen earlier.*

Plan (vi-command-60) has been selected as a plan for the Current Goal (edit-file-effect-1)

No plan failures have been detected.

To edit the file named mayfield, type vi mayfield.

If KIP has more complete information about the particular planning situation, KIP concretes the user's goal to a more specific goal and is able to choose a more specific plan. For example, in (2) if KIP is provided information from the KNOME user model [6,7] that the user is familiar with the emacs editor, the EMACS-COMMAND plan would be considered first. In this case, KIP knows that the user is in the category of EMACS-USERS. Therefore, KIP concretes the user's EDIT-FILE-EFFECT goal to an instance of the EDIT-FILE-WITH-EMACS-EFFECT goal. The plan for the more specific EDIT-FILE-WITH-EMACS-EFFECT goal is the EMACS-COMMAND plan. A KIP trace where of Example (2) with this additional user model knowledge is presented in Figure 6.5.

Figure 6.5: KIP Trace of Plan Determination Using Concretion

User: How do I edit the file named mayfield?

;; In this case, KIP also has information that the user is an emacs user. This
;; information comes from the KNOME user model.

KIP is trying to determine a plan for the list of goals:

```
(edit-file-effect-1
 (Effected-File-27
  (file-1
   (File-Name-101 mayfield-3)))
 (Final-State-Of-Edit-File-Effect-37
  (contents-state-30
   (File-Of-Contents-34 file-1)
   (Contents-Of-Contents-36 file-contents-32)))
 (Initial-State-Of-Edit-File-Effect-53
  (contents-state-44
   (File-Of-Contents-50 file-1)
   (Contents-Of-Contents-52 file-contents-46)))
 (Desired-By-62 uc-user-60))
```

;; This representation of edit-file-effect-1 *is the same as in the previous trace,*
;; except for the additional information that the edit-file-effect-1 *goal is desired*
;; by an emacs-user. This information allows the KIP concretion mechanism to
;; make the following inference:

Concreting edit-file-effect-1 to an instance of edit-file-with-emacs-effect

;; The concretion mechanism has concreted edit-file-effect-1 *because it knows that*
;; emacs-users are generally referring to using emacs when they express an
;; edit-file goal.

Selecting a goal from the List Of Goals ((edit-file-effect-1))

;; The edit-file-effect-1 *goal maintains its name in this trace, even though it is*

;; now an instance of edit-file-with-emacs-effect. Concreted objects maintain
;; their original names, if these names are part of KIP's input. This decision
;; was made in the KIP kodiak implementation in order to make trace output
;; more readable, and for ease in debugging.

```
selecting the remaining goal
 edit-file-effect-1
Looking for a plan for the Current Goal (edit-file-effect-1)
First looking at stored plans
Selected emacs-command as a potential plan
```

;; Notice that only emacs-command is selected as a potential plan,
;; because KIP is planning for a more specific goal.

```
Now specifying the plan for the particular planning situation:
(emacs-command-64
 (File-Arg-Of-Unix-File-Command-67 file-1)
 (Intended-Effect-Of-Emacs-Command-65
  (edit-file-effect-1
   (Effected-File-27
    (file-1
     (File-Name-101 mayfield-3)))
   (Final-State-Of-Edit-File-Effect-37
    (contents-state-30
     (File-Of-Contents-34 file-1)
     (Contents-Of-Contents-36 file-contents-32)))
   (Initial-State-Of-Edit-File-Effect-53
    (contents-state-44
     (File-Of-Contents-50 file-1)
     (Contents-Of-Contents-52 file-contents-46)))
   (Desired-By-Of-Edit-File-With-Emacs-Effect-62 uc-user-60)))
 (Format-Of-Unix-File-Command-71
 (unix-file-command-format-70
  (Format-File-Arg-77 mayfield-3)
  (Command-Arg-75 emacs-string-72))))
No plan failures detected for Candidate Plan (emacs-command-64)
```

To edit the file named mayfield, type emacs mayfield.

6.3 New Plans

There are many situations when KIP does not know of a stored plan for a selected goal. KIP may not have been faced with the situation before. Even in a domain like UNIX, it is impossible to anticipate every potential user goal. Instead of computing a plan from scratch, KIP tries to apply its knowledge of stored plans in order to determine a plan that will be useful in the unique planning situation. KIP does so by attempting to use a stored plan

that addresses a different but related goal. This plan will then be modified in subsequent loops of plan synthesis.

In such a situation, KIP examines its goal hierarchy in order to find the goal which is most similar to KIP's current goal. KIP's plan selection algorithm for new plans is thus called the Goal Similarity Matching Algorithm, or GSMA. This section describes the GSMA algorithm and provides a simple example of its use.

KIP first searches for a goal which is most similar to the current goal by exploiting the hierarchy of goals in the KODIAK network. In order to find the goal in the hierarchy which is most similar to the current goal, KIP looks for the goal which shares more common parents with the current goal than any other goal in the hierarchy. If two or more goals have the same number of common parents, the goal which is the least distant from these parents is chosen as the most similar goal. KIP then examines these plans stored for the similar goal and determines which one of the plans will work in this planning situation. If none of the plans for the similar goal will work for the current goal, KIP tries plans for the next most similar goal.

For example, suppose the user asks the following question:

(3) `How do I move the file george from the machine named renoir to the machine named kim?`

KIP detects the goal of moving the file from the machine renoir to the machine kim. Suppose this is called the MOVE-FILE-TO-DIFFERENT-MACHINE1 goal. This goal is created and placed correctly in the hierarchy by the UC parser and the PAGAN goal analyzer. There is no plan for this individual goal in the KIP knowledge base. KIP then tries unsuccessfully to determine a stored plan for its parent, the MOVE-FILE-TO-DIFFERENT-MACHINE goal. Therefore, KIP uses the GSMA algorithm to search for a plan belonging to the goal most similar to the MOVE-FILE-TO-DIFFERENT-MACHINE goal. KIP does this by finding a goal that shares more common parents with moving a file to another machine than any other goal. Moving a file to another machine is dominated by ethernet (machine-machine links) goals and file-transfer goals. Therefore, KIP first searches for other goals in the hierarchy that are dominated by these two goals. In this case, as shown in Figure 6.6, there is one goal that has both of these parents, the COPY-FILE-TO-DIFFERENT-MACHINE goal. There is one stored plan associated with this goal, the RCP-COMMAND plan.

Therefore, KIP selects the RCP-COMMAND plan as a *potential* plan to move a file to another machine. This plan is then tested during the remainder of the planning process. If the plan does not satisfy all the goals of the user, the plan is modified by determining which simple goals of the user's complex goal are left unsatisfied. Plans for these simple goals will be selected by the same algorithm, i.e., using a stored plan or examining plans of goals similar to these goal parts. In this way, KIP progressively refines the potential plan

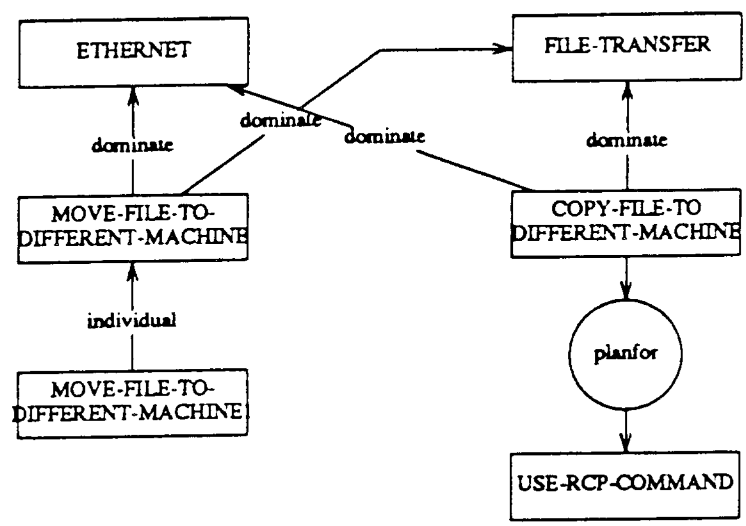

Figure 6.6: Example of Goal Similarity

until as many goals as possible are satisfied. In this case, the RCP-COMMAND plan does not satisfy all the goals of the user. An effect of the RCP-COMMAND plan is that the source file still exists. The user's goal specifies that this file should no longer exist. Therefore, during a subsequent goal detection phase, KIP detects the goal that the source file should be deleted. This goal is passed on to a subsequent plan selection phase.

The GSMA strategy seems cognitively plausible. A human consultant, faced with a problem he has not previously encountered, tries to use solutions which have worked in cases which are similar to the current one. In the next section, in order to more fully understand this algorithm, the GSMA algorithm is contrasted with previous planners' algorithms for plan selection.

6.4 Comparison of GSMA with Previous Approaches

6.4.1 GPS, STRIPS, and ABSTRIPS

It is informative to contrast GSMA with the more traditional way of selecting a potential plan, namely *means-end* analysis [29]. Means-end analysis was used by planners like GPS [10] and STRIPS [12]. It entails examining all the plans in the database and selecting

the plan that reduces the greatest difference between the present state and the goal state. STRIPS is presented with a well-formed formula describing the goal state and the present state as well as a set of formal descriptions of available operations. STRIPS then attempts to prove the truth of the goal state. If an individual subgoal of the goal state cannot be *proven* from the present state, STRIPS selects an operator that will allow the proof attempt to continue.

For example, suppose that STRIPS were asked to find a plan for the goal of moving a file from one machine to another. STRIPS would search for a plan that reduces the greatest difference between the present state of file1 existing on machine1 and not on machine2, and the goal state of file1 existing on machine2 and not on machine1. The differences between the goal state and the present state are:

(1) file1 exists on machine2 in the goal state and file1 does not exist on machine2 in the present state

(2) file1 does not exist on machine1 in the goal state and file1 does exist on machine1 in the present state

STRIPS would search through all its plans and find two pertinent plans: RCP-COMMAND - to copy the file and reduce difference (1), and RM-COMMAND - to delete file1 and reduce difference (2). Since these two plans reduce the same amount of difference, according to the formal criteria of STRIPS, it might arbitrarily choose to first use the RM-COMMAND plan and then look for another plan to reduce the remaining difference. However, since file deletion negates the possibility of using the RCP-COMMAND plan, this plan would fail.

ABSTRIPS [31], which modified STRIPS in order to avoid the consideration of plans that are difficult to achieve, might actually do worse than STRIPS in this particular example. ABSTRIPS used GPS's idea of reducing the *important* differences in the problem first, by assigning criticality levels to differences. ABSTRIPS reduces those differences with the highest criticality first. Criticality levels are assigned by the program itself. These levels are assigned according to the difficulty inherent in satisfying the preconditions for plans to reduce a particular difference. In the cross-machine move example, however, removing a file on machine1 might have a higher criticality level than copying a file from machine1. Copying a file only requires read permission, whereas deleting a file requires write permission on the parent directory. Write permission on a UNIX directory is more difficult to satisfy than read permission on a UNIX file since most UNIX files have read permission.

In GPS, operators are selected according to which operator reduces the greatest difference by using a precomputed difference table. Differences are reduced in order of difficulty according to a predetermined ordering, termed a DIFF-ORDERING. The DIFF-ORDERING is assigned by the GPS implementor based on the ease of task accomplishment.

For example, a human expert might determine that deleting a file is more difficult than creating a file, and assign a DIFF-ORDERING accordingly.

Both ABSTRIPS and GPS could conceivably first choose to reduce difference (2). ABSTRIPS might make this choice because the difference is at a higher criticality level. GPS might make the same decision because the difference is higher up in the DIFF-ORDERING. If these programs solve difference (2) first, they have to deal with the interacting subgoal of the file being deleted before it is copied.

Each of the three planners described above uses means-end analysis, but technical problems prevent them from efficiently determining the proper difference to reduce. STRIPS's formal criteria prohibit it from being able to choose which subgoals to reduce. Both GPS and ABSTRIPS reduce differences according to their order of importance. However, since both use difficulty as a metric for importance, they sometimes erroneously deal with the *difficult* parts of problem before the *important* parts of a problem. When using these planners for knowledge-intensive problems, their greatest limitation is their inability to deal with goal-related knowledge on a sufficiently complex level. Rather than select a plan based on the complex goal of a cross-machine move, these planners are forced to deal individually with the simple goals that comprise this complex goal.

KIP addresses this problem by using the GSMA algorithm to apply knowledge regarding the higher-level complex goals during plan selection. Rather than use only knowledge about the lower-level simple goals, KIP uses the conceptual hierarchy of the complex goals to find plans for similar goals. Since these goals are close in the conceptual hierarchy, they are in fact similar. Thus, a plan for one goal will probably satisfy the other goals to a great degree. Thus, GSMA is viewed as a means of finding the plan that reduces the difference between the user's goal and present state by the greatest amount, where similarity determines which important difference should be reduced. For example, in the cross machine move example, KIP chooses RCP-COMMAND over RM-COMMAND, even though this may not reduce the most difficult difference between the present state and the goal state.

Our limited experience with this Goal Similarity Matching Algorithm suggests that problems of interacting subgoals do not often occur when using the algorithm on knowledge-intensive problems. In addition, using GSMA reduces search time. KIP needs only to consider plans for the few goals that are similar to the user's goals, rather than considering all the possible plans in the database as was necessary in STRIPS and ABSTRIPS. Both these programs use a resolution theorem prover to find the plan that reduces the greatest difference. This theorem prover considers every potential plan in the database.

6.4.2 NOAH

NOAH [32] avoided problems of interacting subgoals by using a least commitment strategy. Unlike STRIPS, NOAH did not specify which order the steps in a plan occurred, until all the plan steps had been selected. NOAH's plan selection strategy is very

similar to the means-end analysis planners described above. NOAH decides on a plan by looking at all the plans in its knowledge base. Like ABSTRIPS, NOAH uses a hierarchical planning approach. Interacting subgoals are usually only discovered when they are in the same hierarchy in the planning structure. If NOAH selects a plan for a subgoal in one hierarchy, this plan may delete preconditions of a plan for another subgoal in a different hierarchy. According to Sacerdoti, due to NOAH's procedural net architecture, it is difficult to detect such problems. If such problems are detected, it is too late to select another plan for either of the subgoals or reorder the plan steps. Instead, the deleted precondition is merely asserted again.

Since KIP uses a declarative representation of plans and goals, such problems are avoided. By using the GSMA algorithm, KIP first determines a plan for the most important part of the complex goal. While constructing plans for less important parts of the complex goal, KIP has access to the preconditions necessary for the important parts of the plan already constructed. Thus, KIP can immediately determine if any new parts of the plan delete preconditions necessary for successful execution of previously specified plan steps. KIP can then make decisions regarding the ordering of plan steps with reference to the important parts of the plan. If a new plan step deletes preconditions necessary for important parts of the plan, KIP can choose to order the new plan step accordingly. For example, KIP might choose to order a new plan step before the plan step which enabled the deleted precondition.

I do not claim that the GSMA algorithm will solve all the interacting subgoals problems NOAH or the other means-ends analysis planners were unable to address, or prevent them from occurring. Instead, I suggest that many of the problems with interacting subgoals occur when plan selection is not done in a knowledge intensive fashion. However, it would be difficult to apply the GSMA algorithm to these planners since they work in the knowledge-deficient domain of the blocks world. The GSMA algorithm is primarily designed for knowledge-intensive domains where the knowledge base will contain similar goals to the planner's current goals. Nonetheless, it is important to compare the GSMA algorithm approach to the plan selection algorithm used by knowledge-deficient planners since these methods are often applied to knowledge-intensive domains.

6.4.3 PLEXUS

In contrast to the means-end analysis planners discussed above, PLEXUS [4] uses an adaptive planning strategy. PLEXUS tries to adapt a pre-existing plan to solve a goal for which no plan is known. PLEXUS differs from KIP in that it focuses on adapting specific complex plans from the same domain or from other planning domains. PLEXUS accomplishes this by attempting to use the steps from an old plan in a new situation. When one of these steps fail, PLEXUS attempts to plan steps in the same hierarchy that will work in the new situation. For example, PLEXUS tries to figure out how to act in the New

York subway system by using a plan which has been developed for the BART (Bay Area Rapid Transit) in San Francisco. PLEXUS realizes that it cannot buy a ticket from a ticket machine, because no machines are present in the subway station. Thus, PLEXUS decides to buy a token from the token seller instead.

Unlike PLEXUS, KIP does not attempt to substitute the actual plan steps of a particular plan which it has selected. If KIP determines that a selected plan will fail in a particular situation, it tries to add to the plan to avoid failure instead of substituting plan steps.

If KIP tried to use plans from other operating system domains, PLEXUS's strategies in plan adaptation could be very useful. However, PLEXUS is limited by the lack of research on the selection of the plan to be adapted. I suggest that the GSMA algorithm could be modified to find such a plan from another domain. Suppose that no successful plan was found for a goal which was in the same domain as the user's goal. The GSMA algorithm could search for a goals which are similar to the user's goal in other plan/goal domains. PLEXUS could then try to adapt a plan for one of these goals. If that plan was not adaptable in the particular planning situation, a plan for a goal in another domain could be adapted instead. Currently, GSMA's simple definition of similarity is based on physical closeness in the hierarchy. In order for GSMA to find similar goals in different plan/goal domains, GSMA's similarity metric would have to be improved so as to detect more complex similarities.

6.5 Conclusion

In this chapter, the two steps of plan determination, plan selection and plan specification, have been discussed. KIP first tries to select a stored plan for the user's goal. If no stored plan is known, KIP selects a plan using its Goal Similarity Matching Algorithm or GSMA. Once KIP has selected a plan, it must be specified for the particular planning situation KIP is considering.

The plan that KIP has selected and specified may not actually work in particular planning situation. A condition of the plan may not be satisfied or the plan may cause a goal conflict with another goal or plan of the user. In the following chapter, KIP's algorithm for the detection of these plan failures is described.

Chapter 7

Plan Failure Detection

7.1 Introduction

Once KIP has determined a potential plan to satisfy the goals of the user, it must decide if the plan will really work in the current planning situation. If the plan will fail in the present state of the world, KIP should either try to modify the plan or determine a new plan for the user.

There are two possible sources of failure that KIP must consider:

(1) *condition failure* -	plan failure due to failure of a condition necessary for successful execution of a plan
(2) *goal conflict failure* -	plan failure due to a conflict between an effect of a plan and a user goal

In plan failure due to condition failure, a plan fails because a condition necessary for successful plan execution is unsatisfied in the present state of the world. When a condition is easy to satisfy, the satisfaction of that condition becomes a new goal for KIP. If a condition is difficult or impossible to satisfy, a new plan must be determined, since the current plan will not work.

Plan failures due to goal conflict occur when the effect of a plan conflicts with a user goal. Before suggesting the plan to the user, KIP must resolve the conflict. KIP may decide to:

(1) modify the plan so that the goal conflict does not occur

(2) determine a new plan that will not cause such a conflict

(3) abandon the goal with which the effect of the plan conflicts

In this chapter, I demonstrate the difficulty of detecting potential plan failures and the characteristics of this problem. In the following chapter the solution used by KIP, involving the notion of concerns, is discussed.

7.2 The Problem: Focusing Attention

The main problem for both condition failure detection and goal conflict detection is the large number of potential failures inherent for any particular plan. Every plan may have many conditions which must be satisfied in order for the plan to execute successfully. Furthermore, the effects of a plan potentially conflict with many goals of the user. These include both user-specified goals and long-term user interests which may be threatened by the plan's effects. It is undesirable for the planner to consider every possible plan failure each time a potential plan is considered. Therefore, a major issue for plan failure detection is focusing the attention of the planner on those conditions and effects which will cause plan failure.

Research in knowledge-deficient planning has not considered the attention-focus problem. The focus of attention on certain conditions in particular has not been an important issue for knowledge-deficient planners. Such planners generally consider only those plans which consist of one or a few simple low-level operations. Since each one of these low-level operations has few conditions, each plan has few conditions which need to be considered. In addition, such planners are only provided with limited knowledge about particular plans. For example, a STRIPS planner might know that a door can only be opened if it is unlocked. However, if the door was nailed shut, STRIPS would not realize that a condition failure had occurred.

For example, let us return to the cross-machine move example discussed in Chapter 6. Once again, suppose that the user asks the following question:

(1) `How do I move the file george from the machine named renoir to the machine named kim?`

As was shown earlier, KIP would select the USE-RCP-COMMAND plan in order to address the goals of the user. KIP must decide if this plan will work in the present planning situation. I now demonstrate the potential plan failures even in the one step USE-RCP-COMMAND plan. These include both condition failures and goal conflict failures.

In order to detect potential failures that might occur if the plan was executed, KIP must consider the conditions necessary for the USE-RCP-COMMAND plan to execute successfully. These include the following conditions:

- the user must have read permission on the source file

- the user must have write permission on the destination file
- the user must have read permission on the directory of the source file
- the user must have write permission on the directory of the destination file
- the user must have an account on the destination machine
- the destination machine must be up
- the user's source machine must be in the destination's machine .rhost file
- the account names on the source and destination machine must be identical

While many of these conditions are unlikely to fail, , KIP must still know about all of the conditions. Such knowledge is needed for those unlikely situations wherein the conditions are not satisfied. The listed conditions are only a fraction of the conditions which KIP stores as necessary for the USE-RCP-COMMAND to execute successfully. KIP's knowledge-base includes many other conditions equally important for successful execution of this plan. Since the USE-RCP-COMMAND plan is described in a plan hierarchy, it inherits other conditions from more general descriptions of UNIX plans in the plan hierarchy. These conditions are even less likely to cause condition failure. They include:

- the user's machine must be up
- the user must have an account on his machine
- the user's terminal must be working

In addition, there are a number of effects of the USE-RCP-COMMAND plan that might conflict with explicit goals or long-term user interests. The problem of detecting goal conflict failures is even more complex than detecting condition failures. Every effect of a plan might potentially conflict with every goal and long-term interest of the user. For example, for the USE-RCP-COMMAND plan, the following are among those effects about which KIP is aware:

- The source file and the destination have the same protection.
- If the user does not have an account on the destination machine, the message is printed: Login incorrect.
- If the destination file exists, it will be deleted.
- The contents of the source file and the destination file are identical.

- If the path is not specified the destination file will be in the user's home directory

These effects might conflict with any explicit user goals about which KIP is aware. The number of explicit user goals depends on the particular interaction with the user. For example, in a long conversation with the user, many explicit user goals might be expressed. The effects of the USE-RCP-COMMAND plan might conflict with any one or more of these explicit goals. Furthermore, any of these effects might conflict with any of the many user interests. These include:

- Keep your password secret.
- Preserve access to your files.
- Don't change other people's files.
- Don't make your directory tree too deep.

Any of these interests might conflict with any of the effects of the USE-RCP-COMMAND plan. Detecting these goal conflicts might therefore entail the comparison of every effect with every goal and interest. Thus, whenever the USE-RCP-COMMAND plan is a potential plan, there are many potential condition failures and goal conflict failures that need to be considered.

Thus, when KIP considers even the relatively simple rcp command in a potential plan, there are many conditions and goal conflicts that need to be examined. Few of these conditions and goal conflicts are likely to cause plan failure. Therefore, KIP should not consider every potential plan failure. In fact, any commonsense planner's algorithm should only consider those plan failures which are most likely.

7.3 Properties of a Plan Failure Detection Algorithm

Three of the properties of a commonsense planner (CSP), discussed in Chapter 1, are particularly important for any CSP algorithm for detecting plan failures. These properties, along with their applicability to plan failure detection are:

(1) *Knowledge Efficiency* -	consider only those potential plan failures which are most likely to occur
(2) *Cognitive Validity* -	attempt to model the method used by human consultants use to detect plan failures
(3) *Uncertainty* -	ability to detect plan failures effectively when the planning situation is not fully specified

Knowledge Efficiency is particularly important for plan failure detection. Commonsense planners generally store many conditions and effects for a particular plan. From a computational perspective, it would be extremely inefficient to check for every potential plan failure, since most of the these potential plan failures are unlikely.

Since commonsense planners attempt to model a human planner, the cognitive validity property entails a further limiting of those conditions which are considered. The failure of a potential plan is usually obvious to a human planner. Rather than considering every possible plan failure, a human planner considers only those conditions which will often cause condition failure. The human planner also considers those effects, goals, and interests which are likely to cause goal conflict.

Finally, a commonsense planner must be able to detect plan failures even when it is *uncertain* of values in the planning situation. Most previous planning research assumed that the values for all the conditions is known. However, in UC, when a user describes a planning problem later passed to KIP, the values for many conditions are usually left out. It would be undesirable to prompt the user for this information, particularly for those values which are not important for the specific planning situation. Instead, a CSP algorithm for plan failure detection should be able to detect plan failures by relying on default situation knowledge. The use of default situation knowledge may entail further processing for all the plan failures considered by a CSP. Therefore, it becomes even more important to limit the conditions, effects, goals, and interests a CSP considers.

The use of default situation knowledge complicates the process of deciding which plan failures should be considered. Since default values depend on the particular planning situation, different plan failures should be considered in different situations. For example, suppose the user asks the following questions:

(2) `User: How do I copy a file named file1?`

(3) `User: How do I copy John's file named file1?`

In (2), a CSP should assume based on its default situation knowledge, that the file belongs to the user. Therefore, there is no reason to check the condition that the file is readable by the user, even though this is one of the many conditions of the UNIX cp command. However, in (3) additional conditions should be checked. Because the file to be copied belongs to someone else, the read permission of the file should be checked.

Therefore, any CSP algorithm for plan failure detection must be able to assess plans wherein only partial knowledge is available. As situations deviate from default situations, different potential plan failures should be considered. In the following chapter, I describe an algorithm for plan failure detection using concerns. One of the important parts of that algorithm is the ability to focus on different potential plan failures when defaults are violated.

7.4 Conclusion

A method of limiting the plan failures which need to be considered is needed. A CSP algorithm that addresses the plan failure detection problem should be able to deal with a large knowledge base that includes the description of the many conditions that are necessary for plans and the many potential sources of goal conflict. A commonsense planner should also have the ability to use default knowledge in the many planning situations wherein complete knowledge is impossible. A CSP should consider only those conditions which are most likely to fail and those goal conflicts which are most likely to occur. The consideration of unlikely plan failures would be inefficient from a processing perspective and would not model human behavior.

One of the major focuses of this chapter has been CSP's dependence on default knowledge for plan failure detection. Any CSP algorithm for plan failure detection must be able to assess plans wherein only partial knowledge is available In the following chapter, we describe an algorithm for plan failure detection using concerns. One of the important parts of that algorithm is the ability to focus on different plan failures when defaults are violated.

This chapter has discussed both types of plan failures detection: condition failure detection, and goal conflict failure detection. Goal conflict failure detection is actually more complex that condition failure detection. A commonsense planner must consider the effects of a plan, the goals and interests of the user, and the goal conflict itself. In Chapter 9, the goal conflict detection problem will be discussed in more detail.

Chapter 8

Concerns

8.1 Introduction

In the previous chapter, the problem of detecting plan failures was discussed. This discussion demonstrated the need for some way to limit the plan failures that are considered during the plan failure detection process. In the present chapter, an algorithm for detecting potential plan failures is described.

In order to address the plan failure detection problem, I introduce a new knowledge structure, termed a *concern*. Concerns identify which aspects of a plan are most likely to cause plan failure. They are part of the knowledge base of the planner reflecting his experience of plan failures. There are two types of concerns that address the two types of plan failures:

(1) *Condition Concerns* -	concerns about the conditions of a plan which are most likely to be unsatisfied
(2) *Goal Conflict Concerns* -	concerns about potential conflicts between effects of a plan and a user goal

Condition concerns refer to those aspects of a plan that are likely to cause plan failure due to a condition of the plan that is needed for successful execution.

Goal conflict concerns refer to those aspects of a plan which are likely to cause plan failure due to a potential goal conflict between an effect of a plan and a goal of the user. Goal conflict concerns relate plans to user goals and to other pieces of knowledge which might be considered extraneous to the plan. Examples of this knowledge include user interests which may be threatened by the plan. Since interests are not usually inferred until such a threat is perceived, goal conflict concerns often refer to conflicts between a potential plan and a long-term interest of the user.

In the current implementation of KIP, there are two kinds of concerns that KIP manipulates:

(1) *stored concerns* - are those concerns stored in KIP's long term knowledge-base

(2) *dynamic concerns* - concerns which arise during the planning process itself

Stored concerns include the important conditions of a stored plan that need to be considered when suggesting a possible plan, and descriptions of potential goal conflicts with the effects of a stored plan. Stored concerns are, therefore, a means for the planner database designer to express his personal experience regarding which aspects of a stored plan are most likely to fail.

Dynamic concerns are usually instances of stored concerns. When a possible plan that is an instance of some previously known stored plan is considered, the stored concerns of the stored plan generate dynamic concerns for the new instance of the plan. Dynamic concerns are introduced by KIP when it notices a potential condition or goal conflict failure, but has not yet decided whether such a failure will occur.

In this chapter, KIP's use of concerns for plan failure detection is described. A simple example of the use of condition concerns is presented, followed by a discussion of the properties of concerns. Then, violated default concerns, which allow KIP to consider potential plan failures in non-default situations, are introduced. Finally, KIP's plan failure detection algorithm using concerns is discussed.

The remainder of this chapter discusses condition concerns, since they are simpler than goal conflict concerns. As noted, at the end of the previous chapter, the goal conflict detection problem is more complex than the condition failure detection problem. Therefore, goal conflict concerns present many more problems than condition concerns. The goal conflict detection problem will be discussed in Chapter 9, and goal conflict concerns will be discussed in Chapter 10. However, most of the discussion of conditions concerns in this chapter also applies to goal conflict concerns. The differences between condition concerns and goal conflict concerns will be discussed in Chapter 10.

8.2 An Example of the Use of Concerns

The simplest use of condition concerns occurs in the case where the user has expressed a goal for which KIP has a stored plan. Let us examine such an example in order to demonstrate how KIP's plan failure detection algorithm uses concerns in order to determine those conditions of a particular plan which should be considered. In this example, experience of the planner knowledge base designer regarding the rcp command

is represented as a concern. His experience is that the rcp command often fails due to a problem with the .rhost file. For example, suppose the user asks the following question:

```
(1) User:  How do I copy the file foo from the
    machine eucalyptus to the machine renoir?
```

KIP is passed the goal of copying the file *foo* from the machines named eucalyptus to the machine named renoir. In this case, KIP's knowledge-base contains a stored plan for the goal of copying a file from one machine to another, namely, the USE-RCP-COMMAND plan.

KIP creates an instance of this plan, which it calls USE-RCP-COMMAND1. KIP must then evaluate the USE-RCP-COMMAND1 plan in order to determine if the plan is appropriate for this particular planning situation. This process entails the examination of those conditions likely to cause plan failure.

In order to examine these conditions, KIP looks at the stored concerns of the stored plan, USE-RCP-COMMAND. For each of the stored concerns associated with the stored plan, it creates a dynamic concern in this individual plan, USE-RCP-COMMAND1. The only stored concern for this individual plan is that the user have the machine named eucalyptus in his .rhost file on renoir. KIP therefore creates a dynamic concern regarding this .rhost file.

KIP then must evaluate the dynamic concern. In this case, there is no explicit information about the .rhost file. Therefore, KIP assumes that .rhost file does not include the source machine. This assumption is based on knowledge about defaults in UNIX. The concern is therefore instantiated as a potential source of plan failure. In a second iteration of plan synthesis, KIP adds to the plan, the step of including eucalyptus in the .rhost file.

Since there are no other stored concerns for this particular plan, KIP assumes that the plan will execute successfully. The plan is then suggested to the user:

```
UC: To copy a file foo to renoir, type rcp foo renoir:. But
    first, add eucalyptus to your .rhost file on renoir.
```

There were many other conditions of the USE-RCP-COMMAND plan that KIP might have considered. Some of these conditions are described in the previous chapter. For example, the condition the file exists is an important condition for the rcp command. However, KIP need not be concerned about this condition in most planning situations, since it is unlikely that this condition will cause plan failure. Hence, such conditions are not stored in KIP's long term memory as stored concerns.

8.3 Properties of Concerns

Concerns allow KIP's CFD algorithm to adhere to the properties for a CFD algorithm described in the previous chapter. Concerns are primarily directed at the knowledge-efficient property of CFD algorithms. Instead of exhaustively searching every potential condition failure, concerns allow KIP to consider only those conditions which are most likely to cause plan failure. Since most plans only have a small number of concerns, the use of concerns limits the amount of processing needed for condition failure detection. KIP is knowledge-efficient both when plan failures are detected and when no failure is detected. KIP detects condition failures quickly by considering only those conditions which are likely to fail. Knowledge-deficient planners would have to proceed through each of the conditions until the detecting failure. Those plans for which KIP does not detect a failure provide an even greater limitation of the processing needed. Since no failure is detected, a knowledge-deficient planner would need to consider every possible condition of such a plan. However, KIP only needs to consider those conditions for which it has stored concerns.

Concerns also attempt to adhere to the cognitive validity property. Plan failures are not as obvious to KIP as they might be to a human consultant. However, the use of concerns adds more to KIP's cognitively validity than the mere limitation of the number of conditions considered. Concerns allow KIP to represent and use a consultant's knowledge that certain conditions in a plan are more likely to cause plan failure than other conditions of a plan. This knowledge could be explained by examining the frequency with which certain conditions are satisfied. However, concerns about a particular plan are at times the result of the consultant's experience in a particular domain.

In the remainder of this section, I describe two different properties of condition concerns:

(1) *degree of concern* -	likelihood that a concern will cause plan failure
(2) *specificity of concern* -	level of generality at which a concern is defined

Degree of concern refers to the likelihood that a particular concern will cause plan failure with respect to a particular plan. This property captures the commonsense notion that some plan failures are more likely to occur than other plan failures. *Specificity of concern* refers to the level of generality in the plan hierarchy which a concern is defined. Some concerns are defined as general concerns of abstract plans that are inherited by more specific plans. Specific concerns can be defined for very specific plans. In this section, I describe these properties of concerns. I also describe how these properties effect KIP's concern algorithm for condition failure detection.

8.3.1 Degree of Concern

Some concerns are more likely to cause plan failure than others. This fact is represented by assigning various degrees of concern to those concerns in the KIP KODIAK knowledge base. In the present implementation of KIP, the decision regarding the degree of the concern is made by knowledge-base designer. The knowledge-base designer must consider two factors (1) the likelihood that a particular condition will not be satisfied, and (2) the likelihood, given a condition is unsatisfied, that plan failure will occur. In KIP, the second factor is generally not considered for condition failure detection. Most of the conditions which have been implemented cause certain failure to the plans for which they are defined.

For example, consider the following three concerns of the USE-LPR-COMMAND plan:

(1) the printer has paper - *high degree*

(2) the printer is online - *moderate degree*

(3) the printer is out of toner - *low degree*

The most likely cause of plan failure involves Concern (1), since the paper runs out quite often, Concern (2) is less likely, and Concern (3) is least likely of all. However, Concern (3) is a more likely source of plan failure than any of the other conditions of the USE-LPR-COMMAND plan that are not cause for concern.

The degree of concern affects KIP's condition concern algorithm in three ways:

(1) *concern consideration order* - the order in which concern are considered

(2) *concern indifference* - deciding which concerns should be ignored

(3) *concern disposal* - concern treatment in the planning process

Degree of concern affects plan failure detection by prescribing an order in which concerns should be considered. Concerns with a high degree of concern are evaluated and considered before concerns with a low degree of concern. This corresponds to a human consultant considering likely problems before considering less likely problems.

For example, suppose the user asks the following question:

```
(2) How do I print out a file?
```

KIP might chose the USE-LPR-COMMAND plan. During plan failure detection, KIP notices the three of USE-LPR-COMMAND's many conditions are cause for concern. Since KIP considers the most likely concerns first, KIP considers concern (1) before considering concern (2).

Concern indifference refers to the decision to ignore certain concerns altogether. KIP ignores all concerns below a threshold level. The threshold can change, depending on the planning situation. For example, when KIP is using a plan not intended for the goal used, a lower threshold value is used. Since KIP is less sure about planning in a new situation, the lower threshold is used. For example, in (2), since the USE-LPR-COMMAND is being used to satisfy the goal for which it was intended, a higher threshold value can be used. In this case, all concerns below a *moderate* degree of concern are ignored. Therefore, KIP ignores concern (3) altogether.

Concern disposal refers to the process of dealing with all the concerns which KIP has identified. Concerns are evaluated in order to determine the degree of concern in the particular planning situation for which KIP is planning. During the evaluation process, a new dynamic concern is created that reflects the stored concern. The dynamic concern is assigned a degree of concern. The degree of the dynamic concern is determined by the degree of the stored concern and the particular planning situation. These concerns are then dealt with according to the degree of the concern. A concern can be elevated to a source of plan failure, disregarded, or overlooked until later in planning process. Concern disposal will be discussed more fully in section 8.8.6.

Degree of concern adds to KIP's knowledge-efficiency and cognitive validity. The evaluation of concerns in a particular order is knowledge-efficient. KIP will generally find plan failures more quickly because it considers the most likely plan failures first. This knowledge-efficient behavior corresponds to a human consultant. Humans, in addition to considering a small number of potential failures, tend to prioritize the order of conditions in accordance with the consultant's expectation of plan failure.

The use of threshold values also adds to KIP's cognitive validity. Thresholds allow KIP to plan more carefully in situations which warrant more caution. The use of higher thresholds in some situations also adds to KIP's knowledge-efficiency. In situations which warrant the use of a high threshold value, KIP might not need to evaluate any of the conditions of a plan.

8.3.2 Specificity of Concern

Concerns are hierarchical in that stored concerns are inherited by a stored plan from the stored plan's parents. KIP can thus have general concerns about a class of plans. These general concerns are defined by descriptions at a high level in the plan hierarchy. These concerns are then inherited by the specific plans in the knowledge base. Such inheritance is automatic in the KODIAK knowledge representation language. Alternatively, very

specific concerns can be defined by the attachment to plans designed specifically for a particular task. Indeed, the specific plan can inherit its entire structure from its parents, except for the concern that is applicable to a particular situation. The ability to access such particular concerns in particular situations corresponds to the human consultant's ability to use general plans for most situations and specific knowledge about plan failures in particular planning situations.

For example, suppose the user asks the following question:

(3) `How do I print a file on the lineprinter in room 508?`

KIP might select the USE-LPR-COMMAND-FOR-508 plan. Since this plan is dominated by USE-LPR-COMMAND, itinherits all of the stored concerns of USE-LPR-COMMAND. These stored concerns include concerns (1), (2), and (3) on page 102. USE-LPR-COMMAND-FOR-508 has the same structure as USE-LPR-COMMAND plan except that the location of the printer is specified as being in room 508. In addition, USE-LPR-COMMAND-FOR-508 has the*specified concern*:

(4) The door of the printer must be securely shut

This concern is specified in the USE-LPR-COMMAND-FOR-508 plan. This specific concern encodes knowledge that the particular condition is a likely source of plan failure in the particular plan. This is termed a specified concern because concern (4) is knowledge-specific to printing a file on the particular lineprinter in room 508.

Specificity of concern allows KIP to adhere to the general-knowledge-application property of commonsense planners. KIP can use knowledge about the concerns of plans which are inherited from a general plan. KIP can also encode specific knowledge applicable to a specific plan about a general concern. For example, suppose that a specific plan inherits concerns from a more general plan. The degree of concern regarding a particular condition in a specific plan might be higher than the degree of concern regarding this condition in a general plan. KIP represents this knowledge by associating a higher degree of concern for this condition in the specific plan.

The fact that conditions of a specific plan can be inherited from a more general plan demonstrates another need for concerns. Due to the hierarchical nature of the database, many conditions far up in the hierarchy would need to be checked if no concern mechanism was included in KIP. In previous implementations of KIP, much time was spent checking many relatively unimportant conditions. For example, every UNIX command has a condition that the machine be up. However, it would be undesirable to check this condition for every possible plan.

8.4 Condition Concern Traces

In this section, a number of KIP traces which focus on condition concerns are presented. In each of these traces, KIP is faced with the same problem:

(4) `How do I print a file on the apple printer?`

The only concern which is above the concern threshold for all three examples is the concern regarding a full paper tray. In Figure 8.1, KIP knows that the paper tray is empty. In Figure 8.2, KIP knows that the paper tray is full. And in Figure 8.3, no information regarding the paper tray has been provided to KIP in the plan description. Therefore, KIP must use default knowledge. The goals and plan are fully described in Figure 8.1, but are abbreviated in Figures 8.2 and 8.3.

Figure 8.1: KIP Trace of Apple Printer Printing with an Empty Paper Tray

User: How do I print a file on the apple printer?

;; In this example, KIP also has the input that the paper-tray of this
;; particular printer is empty:

```
(ap-4
 (Paper-Tray-Status-Of-Apple-Printer-154 empty-32))
```

Entering Goal Establishment Phase:

PAGAN produces:

```
(pfe-1
 (Destination-Of-Apple-Print-File-Effect-30 ap-4)
 (File-Arg-Of-Print-File-Effect-51
  (file-46
   (Contents-55
    (file-contents-54
     (Printing-Of-57 printing-52)))
   (File-Name-112 file-name-string-107)))
 (Output-Printed-On-53 printing-52)
 (Initial-Value-Of-Print-File-Effect-105 blank-106)
 (Final-State-Of-Print-File-Effect-67
  (printed-on-state-60
   (Value-of-Printed-On-66 printing-52)
   (Object-Of-Printed-On-64 paper-61)))
 (Initial-State-Of-Print-File-Effect-103
  (printed-on-state-74
   (Value-of-Printed-On-102 blank-106)
   (Object-Of-Printed-On-100 paper-61)))
 (Output-73 paper-61)
```

```
(State-Change-Interval-71 state-change-time-70))
;; KIP has received the user's goal (pfe-1) of printing on the apple printer
;; from the PAGAN goal analyzer. Also, KIP has knowledge about the
;; particular printer that the user wants to print on, i.e. ap-4
;; The final state is that some printing appear on the paper
;; The initial state is that the paper is blank
;; The printing: printing-52 is the printing of the contents of the file
KIP is trying to determine a plan for the list of goals:
(pfe-1)
Selecting a goal from the List Of Goals ((pfe-1))
selecting the remaining goal
 pfe-1
Looking for a plan for the Current Goal (pfe-1)
Entering Plan Determination Phase:
First looking at stored plans
Selected lpr-pap-command as a potential plan
Now specifying the plan for the particular planning situation:
(lpr-pap-command-34
 (Printer-Name-Argument-41 ap-string-123)
 (Destination-Of-Lpr-Command-37
  (ap-4
   (Paper-Tray-Status-Of-Apple-Printer-33 empty-32)
   (Printer-Abbreviation-Of-Apple-Printer-121 ap-string-123)))
 ;;  The destination of the printing is a particular apple printer called ap-4
 ;;  Its paper tray is empty, and its printer-abbreviation is ap
 (Format-Of-Lpr-Command-45
  (unix-file-printing-command-format-44
   (Command-Arg-116 lpr-string-113)
   (Printer-Arg-120 ap-string-123)
   (Format-File-Arg-110 file-name-string-107)))
 ;; The user could type this format as lpr -Pap filename
 (Intended-Effect-Of-Lpr-Pap-Command-35 pfe-1)
 ;; pfe-1 is expanded above
 (File-Arg-Of-Unix-File-Command-47 file-46)
 (Intended-Effect-Interval-125 state-change-time-70)
 (Command-Interval-127 interval-time-126))
Entering Plan Failure Detection Phase:
Evaluating the condition concern: out-of-paper-concern-130
The condition of concern is the paper-tray-status-of-apple-printer-state-160 of the
 ap-4
The current value of the condition is empty-32
The desired value is full
The current value cannot be the desired value, so a high level of concern is returned
;; KIP chooses a high value of concern, because the out-of-paper-concern has a
;; high degree of concern, and according to KIP's input the desired value is
;; mutually-exclusive with respect ot the current value.
```

Entering Goal Establishment Phase:

Creating a goal that reflects a change from the Current Value (empty-32)
to the Desired Value (full)

Creating the goal:

```
(have-full-paper-tray-effect-226
 (Experiencer-Of-State-Change-174 ap-4)
 (Final-Value-Of-Have-Full-Paper-Tray-Effect-246 full-252)
 (Initial-Value-172 empty-32)
 (Final-State-Of-Have-Full-Paper-Tray-Effect-224
  (paper-tray-status-of-apple-printer-state-220
   (Printer-Of-Paper-Tray-Status-214 ap-4)
   (Apple-Printer-Paper-Tray-Status-221 full-252)))
 (Initial-State-Of-Have-Full-Paper-Tray-Effect-227
  (paper-tray-status-of-apple-printer-state-160
   (Printer-Of-Paper-Tray-Status-155 ap-4)
   (Apple-Printer-Paper-Tray-Status-161 empty-32))))
```

;; KIP has created the goal regarding the ap-4 apple printer
;; The final-state is that the paper-tray is full
;; The initial-state is that the paper-tray is empty

Asserting the fact that the final-state of the goal (paper-tray-status-of-apple-printer-state-220)
starts before the start of plan interval (lpr-pap-command-34)
So that the condition holds before the plan is executed

;; KIP knows that conditions of a plan must be satisfied before the plan is executed

Selecting a goal from the List Of Goals ((have-full-paper-tray-effect-226))
selecting the remaining goal
have-full-paper-tray-effect-226

;; KIP is reiterating on its plan synthesis algorithm on the goals generated by concerns

Looking for a plan for the Current Goal (have-full-paper-tray-effect-226)
First looking at stored plans
Selected fill-paper-tray as a potential plan

Now specifying the plan for the particular planning situation:
(fill-paper-tray-264
 (Intended-Effect-Of-Fill-Paper-Tray-265 have-full-paper-tray-effect-226))
Asserting that fill-paper-tray-264 comes before lpr-pap-command-34

;; KIP makes this assertion because it knows that the intended-effect of the
;; fill-paper-tray-264 *plan is a condition of the* lpr-pap-command-34 *plan.*

No plan failures detected for Candidate Plan (fill-paper-tray-264)

To print the file named foo on the apple printer, type lpr -Pap foo.
But first, fill the printer with paper.

Figure 8.2: KIP Trace of Apple Printer Printing with a Full Paper Tray

User: How do I print a file on the apple printer?

;; In this case, KIP has the same goal as in the previous example, except KIP also
;; has the additional informatation regarding the ap-4 *apple printer:*

(ap-4
(Paper-Tray-Status-Of-Apple-Printer-154 full-32))

;; KIP now goes through the plan steps as in the previous example
;; This trace has been abbreviated so as to highlight the differences between this
;; example and the example in Figure 8.1 on page 105.

KIP is trying to determine a plan for the list of goals:
(pfe-1)
Selecting a goal from the List Of Goals ((pfe-1))
selecting the remaining goal
pfe-1
Looking for a plan for the Current Goal (pfe-1)
First looking at stored plans
Selected lpr-pap-command as a potential plan
Now specifying the plan for the particular planning situation:
(lpr-pap-command-34)
Evaluating the condition concern: out-of-paper-concern-130
The condition of concern is the paper-tray-status-of-apple-printer-state-160 of the
ap-4
The current value of the condition is full-32
The desired value is full
Since the desired value dominates the current value, there is no need for concern

To print the file named foo on the apple printer, type lpr -Pap foo.

;; KIP has suggested the same initial plan to the user, but has not told the user
;; that he needs to fill the paper tray.

Figure 8.3: KIP Trace of Apple Printer Printing with an Unknown Paper Tray

User: How do I print a file on the apple printer?

;; KIP has detected the same goal as in the previous two traces. However, KIP has
;; no information regarding the paper tray.
;; This trace has been abbreviated so as to highlight the differences between this
;; example and the examples in Figure 8.1 and Figure 8.2.

KIP is trying to determine a plan for the list of goals:
(pfe-1)
Selecting a goal from the List Of Goals ((pfe-1))
selecting the remaining goal

pfe-1
Looking for a plan for the Current Goal (pfe-1)
First looking at stored plans
Selected lpr-pap-command as a potential plan
Now specifying the plan for the particular planning situation:
(lpr-pap-command-26)
Evaluating the condition concern: out-of-paper-concern-86
The condition of concern is the paper-tray-status-of-apple-printer-state-110 of the apple-printer-22
The current value of the condition is empty-or-full-92
;; empty-or-full is the superordinate category of empty and full, and is generated
;; as a filler that can be filled by either empty or full.
The desired value is full
Current Value (empty-or-full-92) is less specific than the Desired Value (full)
Try to make the current value more specific using defaults
;; Since KIP does not have a specific value it must use default knowledge to
;; determine the most likely value for the condition
Default-value is empty.
The Default Value (empty) is mutually exclusive with respect to the Desired Value (full)
Since the Default Value (empty) is more specific than the Current Value (empty-or-full-92), instantiate concern that default value be changed
;; If the default value was not more informative than the current value, the current
;; value could be used. In this way, KIP does not need to consider the uncertainty
;; of the default knowledge. In this case, the default value has a moderate degree
;; of certainty.
Creating a goal that reflects a change from the Current Value (empty) to the Desired Value (full)
Creating the goal:
(have-full-paper-tray-effect-155)
Asserting the fact that the final-state of the goal (paper-tray-status-of-apple-printer-state-149) starts before the start of plan interval (lpr-pap-command-26)
So that the condition holds before the plan is executed
Selecting a goal from the List Of Goals ((have-full-paper-tray-effect-155))
selecting the remaining goal
have-full-paper-tray-effect-155
Looking for a plan for the Current Goal (have-full-paper-tray-effect-155)
First looking at stored plans
Selected fill-paper-tray as a potential plan
Now specifying the plan for the particular planning situation:
(fill-paper-tray-187
(Intended-Effect-Of-Fill-Paper-Tray-188 have-full-paper-tray-effect-155))
Asserting that fill-paper-tray-187 comes before lpr-pap-command-26
No plan failures detected for Candidate Plan (fill-paper-tray-187)
To print the file named foo on the apple printer, type lpr -Pap foo.
But first, fill the printer with paper.

;; KIP has generated the same plan as in the first trace, but in this case used
;; default knowledge.

8.5 Default Concerns vs. Violated Default Concerns

In the description of the plan failure detection problem, the importance of plan failure detection in non-default situations was discussed. I demonstrated that any algorithm for plan failure detection must be able to detect plan failures in such situations. I now introduce a new type of concern to address the condition failure detection problem, termed *violated default concerns*. This new type of concern allows KIP to identify potential plan failures in non-default situations. Moreover, violated default concerns drive the plan failure detection process. In this section, I describe violated default concerns and their impact on the plan failure detection process. In the following section, I describe KIP's complete algorithm for detecting plan failures due to condition failure. Violated default concerns are a very important part of this detection process.

The previous examples of concerns have all been *default condition concerns*, applicable given a certain set of default assumptions about the world. When KIP is asked to determine a plan in a unique situation where such defaults are violated, violated default concerns are also considered. These new concerns allow KIP to consider potential plan failures that would be ignored when defaults are not violated.

Violated default concerns are implemented in KIP by attaching concerns to categories of plans in non-default situations. When a default is violated, KIP looks at the category of plans that addresses the particular violated default. KIP then matches the concerns of that plan category against the conditions of the plan under consideration.

For example, suppose the user asks the following questions:

(5) `How do I print out a file?`

(6) `How do I print out John's file named test1?`

In (5), KIP chooses the USE-LPR-COMMAND. The stored *default concerns* for this plan are that the printer is up, and the printer has enough paper. They are *default concerns* since they are KIP's concerns in default situations. Another condition of the USE-LPR-COMMAND is that the user has read permission on the file he is trying to print. However, since the default is that the user has read permission on most files in the system, it is unlikely that this condition will cause plan failure. The designer of the planner knowledge-base has experience that this plan seldom fails due to the failure of this condition. Therefore, the

stored plan does not include a stored *default concern* for this condition. Since no defaults have been violated, no violated default concerns are considered.

However, when KIP relies solely upon these *default concerns*, if KIP chooses the USE-LPR-COMMAND plan for (6), KIP would have exactly the same concerns as (5). However, KIP should have additional concerns because of new information in the planning situation. In (6), KIP should become concerned about read permission. The problem specifies that the file is John's file. Although it is likely that the user has read permission on most files in the system, it is less likely that he will have read permission on another user's personal files. This concern should be considered by KIP, even though read permission is not mentioned as one of the stored *default concerns* for this particular plan.

In (6), KIP realizes that John's ownership of the file violates the default of the file belonging to the user. As the file belongs to someone else, a few concerns arise, e.g. the file being readable and the file being writable. These violated default concerns are stored in the category of plans that manipulate other user's files. These concerns are generally applicable whenever the user is manipulating others' files. Such violated default concerns are then matched with the conditions of the UNIX-printing plan, the lpr command. In (6), only the concern that the file be readable is applicable, since the other concerns are not conditions of the lpr command. A computer trace of Example (6) is presented in Figure 8.4 on page 112.

A CSP algorithm for condition failure detection must be able to distinguish between those aspects of a particular planning situation which require the consideration of additional conditions and those aspects of a plan which should not lead to further consideration of additional conditions. KIP makes these distinctions by deciding which parts of a planning problem violate defaults that are usually assumed by KIP. When such a default is violated, a new violated default concern is considered, which may result in consideration of additional conditions. Other aspects of a plan do not require such additional processing.

For example, in (6), the user has specified the goal of printing someone else's file. This goal violates KIP's default assumption that the user generally manipulates his own files. Therefore, a violated default concern regarding read permission should be introduced in (6). However, the user has also specified that the name of the file to be printed is test1. However, the fact that the file's name is test1 does not violate any defaults, since KIP has no information about the default names of files. Therefore, no violated default concern is created and no other conditions need to be considered as a result of this information.

Violated default concerns have the same properties as those of default concerns discussed in the previous section. Violated default concerns are often defined on a abstract level, e.g. the manipulation of other's files. These concerns are related directly to the definition of the particular default. However, there can also be more specific violated default concerns attached to more specific plans. KIP always considers the most specific violated default concerns in the plan hierarchy.

In addition, violated default concerns can have varying degrees of concern. This

is particularly important in regard to violated default concerns defined at a general level. As was discussed in reference to default concerns, concerns with a higher degree of concern are evaluated first. Concerns with a degree of concern below a threshold level may not be considered at all.

A powerful tool for defining violated default concerns is to mix both the degree and specificity properties. Specific violated default concerns can be attached to more specific descriptions of plans with a degree of concern that is different from the degree defined at a higher level of abstraction. For example, a specific violated default concern could be defined for a particular plan with a high degree of concern. The low degree of concern associated with the abstract plan will be ignored and the specific violated default concern will be evaluated.

The use of violated default and default concerns is primarily designed to address the uncertainty property of CSP algorithms. KIP may not be aware of the values of many plan conditions. Instead, as shown in the previous chapter, KIP must rely on a large body of default knowledge. Unless given information to the contrary, KIP relies on defaults being true and only considers default concerns. When KIP has information that a default has been violated, it considers violated default concerns. Therefore, KIP does not have to prompt the user for information about the particular planning situation, but relies on its default knowledge.

Thus, default/violated-default concerns allow KIP to deal with a large body of default knowledge. Concerns circumvent the need to refer to default knowledge regarding the value of those conditions about which KIP is unconcerned. Thus, concerns provide a way of dealing with a large body of default knowledge by allowing the planner to compute only the default values of conditions of concern to KIP. The use of violated default concerns also conforms to the cognitive validity property of commonsense planners. Violated default concerns allow KIP to deal with planning situations in which user-specified properties do not conform with the default assumed by the system. This approach seems cognitively correct. A human consultant, when given a problem, immediately notices those aspects of a problem that differ from the norm. Furthermore, a human consultant is able to differentiate those aspects of a problem that merely differ from the norm from those aspects which could cause plan failure. Violated default concerns allow KIP to both store (stored concerns) and process (dynamic concerns) those potential plan failures that might occur when a planning situation does not conform to default knowledge.

A computer trace of Example (6) is presented in Figure 8.4 below.

Figure 8.4: KIP Trace of a Violated Default Concern

User: How do I print out John's file named test1?

Entering Goal Establishment Phase:

(print-file-effect-1

```
 (File-Arg-Of-Print-File-Effect-28
  (file-2
   (Owner-26 john-3)
;;; The owner of the file is specified as being john. PAGAN knows that john is
;;; different from the uc-user. PAGAN believes that the user would not specify the
;;; fact that the file belongs to someone unless the file belongs to someone else. The
;;; fact that the owner of the file is not the uc-user causes a violated default later in
;;; this trace.
   (Contents-32
    (file-contents-31
     (Printing-Of-34 printing-29)))
   (File-Name-24 test1-1)))
 (Output-Printed-On-30 printing-29)
 (Initial-Value-Of-Print-File-Effect-58 blank-61)
 (Final-State-Of-Print-File-Effect-43
  (printed-on-state-35
   (Value-Of-Printed-On-63 printing-29)
   (Object-Of-Printed-On-39 paper-36))
 (Initial-State-Of-Print-File-Effect-56
  (printed-on-state-48
   (Value-Of-Printed-On-60 blank-61)
   (Object-Of-Printed-On-52 paper-36)))
 (Output-47 paper-36)))
Selecting a goal from the List Of Goals ((print-file-effect-1))
selecting the remaining goal
 print-file-effect-1
Entering Plan Determination Phase:
Looking for a plan for the Current Goal (print-file-effect-1)
First looking at stored plans
Selected lpr-command as a potential plan
Now specifying the plan for the particular planning situation:
(lpr-command-64
 (Printer-Name-Argument-69 printer-abbreviation-string-68)
 (Destination-Of-Lpr-Command-67
  (printer-66
   (Printer-Abbreviation-71 printer-abbreviation-string-68)))
;; The user has not specified which printer, he wants to print on, so the
;; printer-abbreviation is unknown. The user could use a number of different
;; printers that are at his disposal. If the user had referred to a particular printer,
;; e.g. the apple printer, as in the example on page 51, PAGAN
;; would resolve the reference as to which printer the user is
;; referring.
 (Owner-Of-File-Arg-77 john-3)
;; This is the relation whose default is actually violated. This
;; relation is equated to the relation path (file-arg owner).
```

```
(Format-Of-Lpr-Command-73
 (unix-file-printing-command-format-72
  (Command-Arg-87 lpr-string-84)
  (Printer-Arg-89 printer-abbreviation-string-68)
  (Format-File-Arg-81 test1-1)))
 (Command-Name-Of-Lpr-Command-85 lpr-string-84)
 (Intended-Effect-Of-Lpr-Command-65 print-file-effect-1)
 ;; This is the print-file-effect-1 as is specified above.
 (Actor-Of-Unix-Command-96 uc-user-1)
 (File-Arg-Of-Unix-File-Command-75 file-2))
```

Entering Plan Failure Detection Phase:

The Current Value (john-3) cannot be the Desired Value (uc-user),
so a violated default is detected

;; KIP knows that the owner-of-file-arg relation is usually a uc-user. In most
;; problems the owner-of-file-arg is not specified, and KIP assumes that it has the
;; default value. In this case, the owner-of-file-arg is john-3, *which KIP knows is not*
;; the uc-user. Therefore, a violated default is detected.

This default is reprensented by:

```
(owner-of-file-arg-state
 (Object-Of-Owner-Of-File-Arg unix-file-command)
 (Value-Of-Owner-Of-File-Arg user)
 (Default-Value-Of-Owner-Of-File-Arg-State uc-user)
```

;; In order to store the default, a owner-of-file-arg-state absolute is stored in
;; the knowledge base. Its object is the unix-file-command, and its value is a
;; user. The default-value is the uc-user.

```
 (Default-Violation-Concern-Of-Owner-Of-File-Arg-State
        not-owner-violated-default-concern))
```

The Violated Concern (not-owner-violated-default-concern) is detected

;; The representation of owner-of-file-arg-state stores the fact if this default is
;; violated, KIP should instantiate the not-owner-violated-default-concern. This is a
;; general violated default concern that applies to many commands. However,
;; a particular unix-file-command may have one or more specific concerns that are
;; inherited from this concern.

Therefore the Individual Violated Default Concern (no-read-permission-of-file-arg-concern-106)
is instantiated

;; In this case, one particular violated default concern is detected. This is the
;; concern that the user have read-permission on the file. This concern is actually
;; stored for all unix-file-command, since they all need read-permission on the file
;; file-arg. Write permission is only a violated-default concern for those command
;; which actually effect the file.

Evaluating the condition concern: no-read-permission-of-file-arg-concern-106
The condition of concern is the read-permission-state-110 of the file-2
The current value of the condition is truth-value-112
The desired value is true

;; KIP is evaluating the concern in this particular planning situation

Current Value (truth-value-112) is less specific than the Desired Value (true)
Try to make the current value more specific using defaults
Default-value is anything.
Since the Current Value (truth-value-112) is more specific than the Default Value (anything),
Instantiate concern that current value be changed

Creating a goal that reflects a change from the Current Value (truth-value-112)
to the Desired Value (true)

;; This goal is generated due to the violated default concern

```
Creating the goal:
(add-read-permission-of-file-effect-145
 (Final-Value-Of-Add-Read-Permission-Of-File-Effect-166 true-168)
 (Initial-Value-Of-Normal-State-Change-146 truth-value-112)
 (Final-State-Of-Add-Read-Permission-Of-File-Effect-150
  (read-permission-state-160
   (Value-Of-Read-Permission-175 true-168)
   (Object-Of-Read-Permission-161
    (file-2
     (Owner-26 john-3)
     (Contents-32
      (file-contents-31
       (Printing-Of-34 printing-29)))
     (File-Name-83 test1-1)))))
 (Initial-State-Of-Add-Read-Permission-Of-File-Effect-143
  (read-permission-state-110
   (Value-Of-Read-Permission-124 truth-value-112)
   (Object-Of-Read-Permission-114 file-2)))
 (Effected-File-148 file-2))
```

Asserting the fact that the final-state of the goal (read-permission-state-160)
starts before the start of plan interval (lpr-command-64)
So that the condition holds before the plan is executed

Entering Goal Establishment Phase:

Selecting a goal from the List Of Goals ((add-read-permission-of-file-effect-145))
selecting the remaining goal
add-read-permission-of-file-effect-145

Entering Plan Determination Phase:

Looking for a plan for the Current Goal (add-read-permission-of-file-effect-145)

First looking at stored plans
No stored plans found

Trying to determine a new plan
Looking for a plan for the Current Goal (add-read-permission-of-file-effect-145)
among goals which are similar to the current goal

;; KIP is unsuccessful at finding a plan for this goal, because there is no way of
;; changing the read permission of a file which is owned by someone else

Trying to decompose the Current Goal (add-read-permission-of-file-effect-145)

No decomposition found
There is no plan for the current goal
Adding the concern generated goal to the list of unaddressed concerns
Kip has not been able to address these goals in the
List Of Unaddressed Concerns (add-read-permission-of-file-effect-145)
These concerns will be expressed to the user.
To print the file named test1, type lpr test1.
However, this plan will not work if the file is read protected.

8.6 KIP's Algorithm for Dealing with Condition Concerns

There are three main parts of KIP's goal conflict concern algorithm:

(1) Concern Retrieval - retrieve the concerns from KIP's planning knowledge base

(2) Concern Evaluation - evaluate the concerns in the particular planning situation

(3) Concern Treatment - decide how the planning process should proceed based on the concern information

First, the three parts of this process are described. I then describe some of the implementation and representation details.

In the Figure 8.5, I have expanded on those parts of the KIP's planning algorithm in which condition concerns play an important role.

8.6.1 Concern Retrieval

After KIP detects the goals of the user, it selects a potential plan and creates an instance of that plan. KIP then checks for any violated defaults in the particular planning situation by comparing the values of properties in the planning situation, that have been specified by the user, against the default values for those properties. For each violated default, KIP determines the most specific stored violated default concerns. Since some violated defaults may generate concerns which are not conditions of the present planning situation, KIP matches all the violated default concerns against the conditions of the plan. KIP discards all concerns which are not conditions of the plan.

KIP then gathers all the default stored concerns for the plan. Once both the default concerns and violated default concerns are gathered, it sorts them based on the degree of

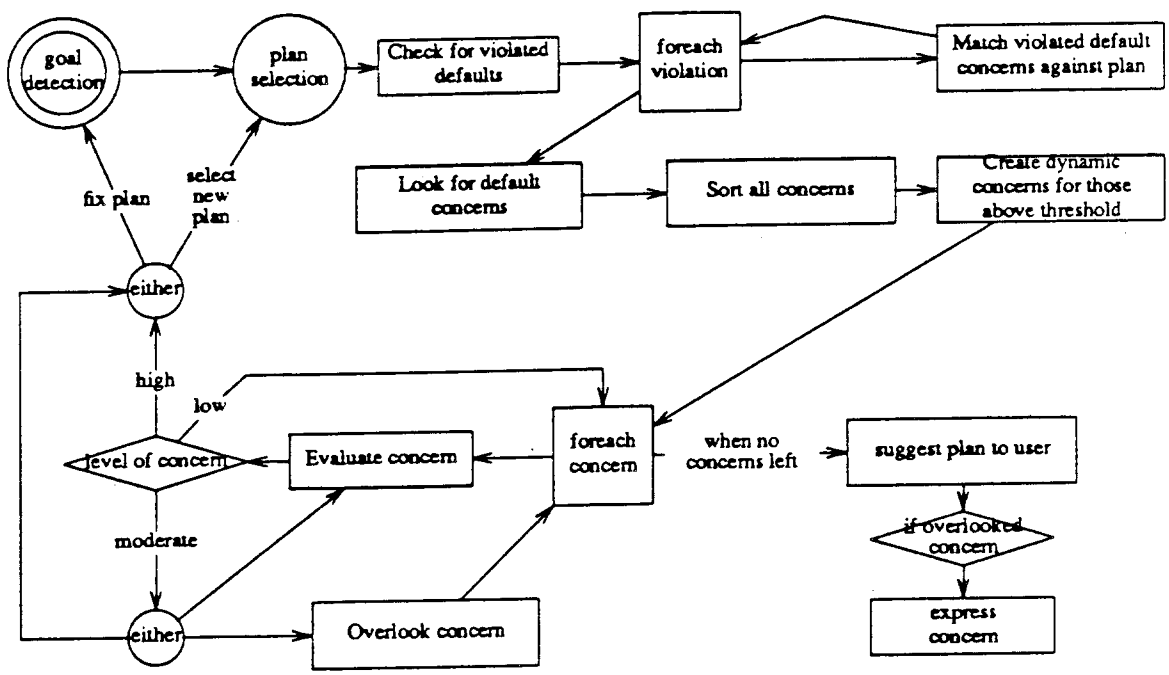

Figure 8.5: KIP's Concern Algorithm

concern. KIP then decides on a threshold level for concern, based on the planning situation. For example, if the plan is the normal plan for these goals, a high threshold will be chosen. A lower threshold is chosen when the plan has not been used before. Concerns below the threshold level are discarded.

8.6.2 Concern Evaluation

KIP then creates dynamic concerns for each of the stored concerns. It evaluates these dynamic concerns in the order of the degree of stored concern. KIP evaluates the condition concerns by determining whether conditions are true in KIP's model of the world. During this evaluation, KIP assigns a new degree of concern to the dynamic concern based on the particular planning situation. Conditions which are known to be true are assigned a low degree of concern, and conditions that are known to be false are assigned a high degree of concern.

However, many of the values will not be known and must be provided from uncertain default knowledge. Therefore, the computed degree of concern of the dynamic concern is calculated by evaluating both the degree of concern of the stored concern and the degree of certainty in the default knowledge. If the degree of certainty of the default knowledge is high, then the computed degree of concern is the same as it would be if the default value was supplied in the input. If the degree of certainty is moderate or low, the computed degree of concern is correspondingly lower. For example, suppose KIP evaluates a dynamic concern whose stored concern has a high degree of concern. The default knowledge claims that the condition is unlikely but still possible. In this case, KIP would decide that the degree of concern of the dynamic concern is moderate.

8.6.3 Concern Treatment in the Planning Process

Once KIP has evaluated a concern it can proceed in one of three ways, depending on the degree of the particular concern. If the degree of concern is *low*, KIP can choose to *disregard* the concern. If the concern is disregarded it is no longer considered at all. KIP can try to modify other parts of the plan, or suggest the plan to the user with no reservations.

If the degree of concern is *high*, KIP can choose to *elevate* the concern to a source of plan failure. In this case, KIP determines that it is very likely that the plan will fail. KIP tries to modify the plan in order to change the value of this condition. Alternatively, KIP tries to find another plan.

The most complex case is when the degree of concern is *moderate*. In this case, KIP can choose to *disregard* the concern, or *elevate* it to a source of plan failure. KIP can also choose to *overlook* the concern. Once KIP has developed a complete plan for this problem, it is once again faced with the need to deal with the *overlooked* concern. If the plan will work, except for the overlooked concern, KIP can again choose to *disregard* the

concern, or *elevate* it to a source of plan failure. At this point, KIP can also choose to suggest an answer to the user. Depending on the degree of the overlooked concern, KIP may choose to express the concern to the user in the answer.

8.6.4 Implementation and Representation

Stored condition concerns are presently implemented by creating a different CONCERN concept for each concern. Also, a Concern-Of relation is added between each concern and the plan for which there is cause for concern. Degrees of concern are represented by creating a Concern-Level relation between the particular concern and a degree. Degrees are presently implemented as numbers from one to ten. Dynamic condition concerns are implemented as instances of stored concerns. Condition concerns have a Desired-Value relation which refers to the value that the Concern-Condition should have. If the Desired-Value does not hold, a goal is instantiated to reflect the condition concern.

For example, in Figure 8.6, the OUT-OF-PAPER-CONCERN is represented. This concern is a Concern-Of the LPR-PAP-COMMAND. The Concern-Object is a printer, which must also be the Destination of the LPR-PAP-COMMAND about which there is concern. The Concern-Condition is a PAPER-TRAY-STATE, where the object must be the Destination of the LPR-PAP-COMMAND. The degree of concern is high, represented by the number 9.

Defaults are implemented in the current version of KIP by attaching the default values of a relation of a plan to a state which reflects the relation. For example, in Figure 8.7, UNIX-FILE-COMMAND has a Owner-Of-File-Arg relation. The OWNER-OF-FILE-ARG-STATE absolute is created to reflect the Owner-Of-File-Arg relation. The Default-Value of OWNER-OF-FILE-ARG-STATE is UC-USER, i.e. the user of the UC system. When KIP notices that this default is violated, a violated default concern is instantiated. Context dependent defaults are implemented by exploiting the concretion mechanism of UC, which tries to find the most specific concept in the hierarchy. Since KIP retrieves the most specific plan in the knowledge-base, it automatically retrieves the most specific defaults.

Violated default concerns are implemented by creating a different VIOLATED-DEFAULT-CONCERN concept for each violated default concern. A Default-Violation-Concern relation is added between the class of violated default concerns and the stored default which is violated. Therefore, when KIP has found the default that has been violated, it looks for members of the class of violated default concerns referenced by this default that are relevant to the potential plan. For example, Figure 8.8 shows a Default-Violation-Concern relation between the OWNER-OF-FILE-ARG-STATE and NOT-OWNER-VIOLATED-DEFAULT-CONCERN.

When the default of OWNER-OF-FILE-ARG-STATE is violated, KIP determines if its potential plan is among the class of plans that have a NOT-OWNER-VIOLATED-DEFAULT-CONCERN concern. Suppose that KIP is considering the LS-COMMAND-ON-SUN plan discussed in Figure 1.1 on page 9. The LS-COMMAND-ON-SUN plan has a violated default

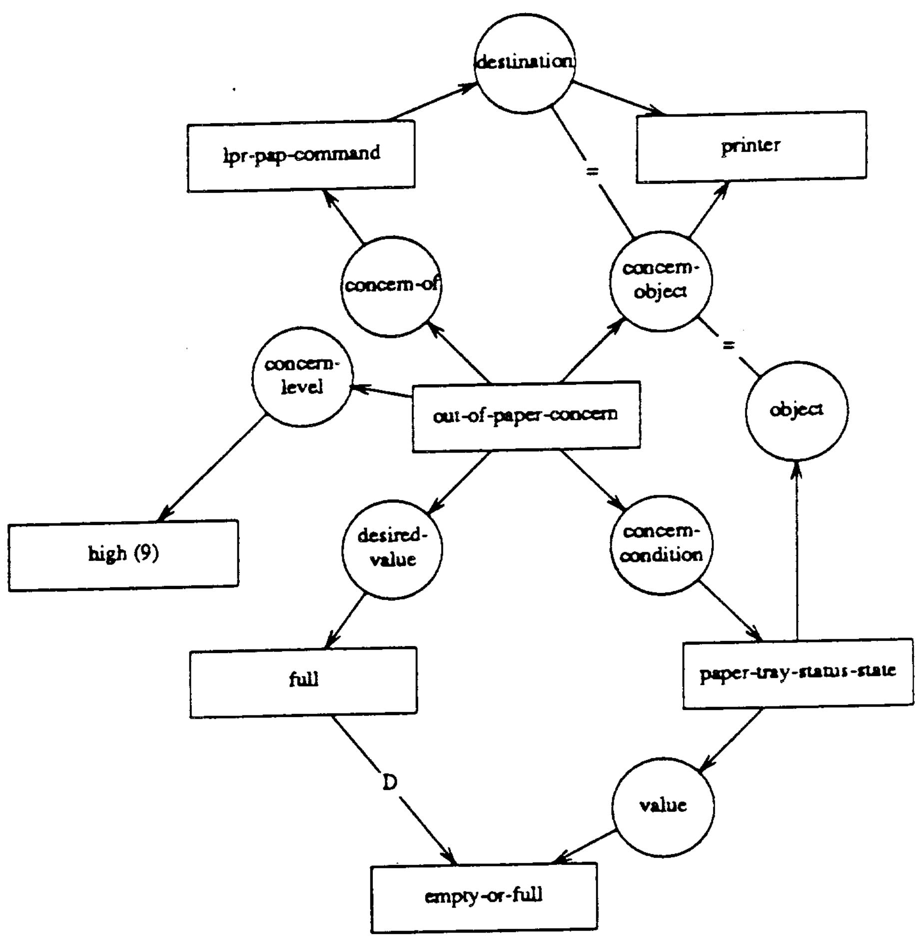

Figure 8.6: Representation of OUT-OF-PAPER-CONCERN

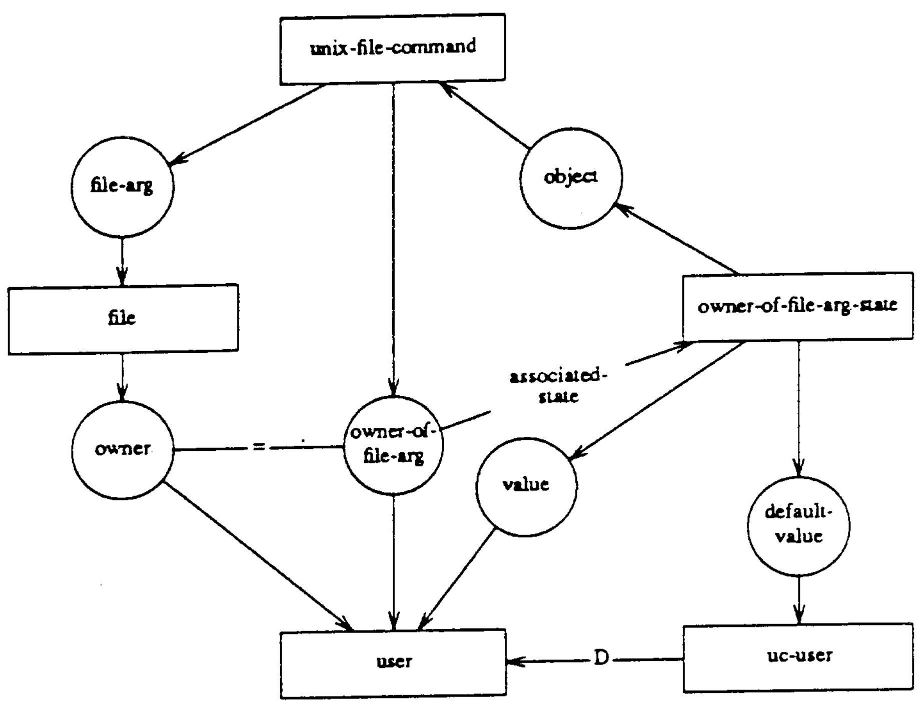

Figure 8.7: Representation of OWNER-OF-FILE-ARG-STATE Default Knowledge

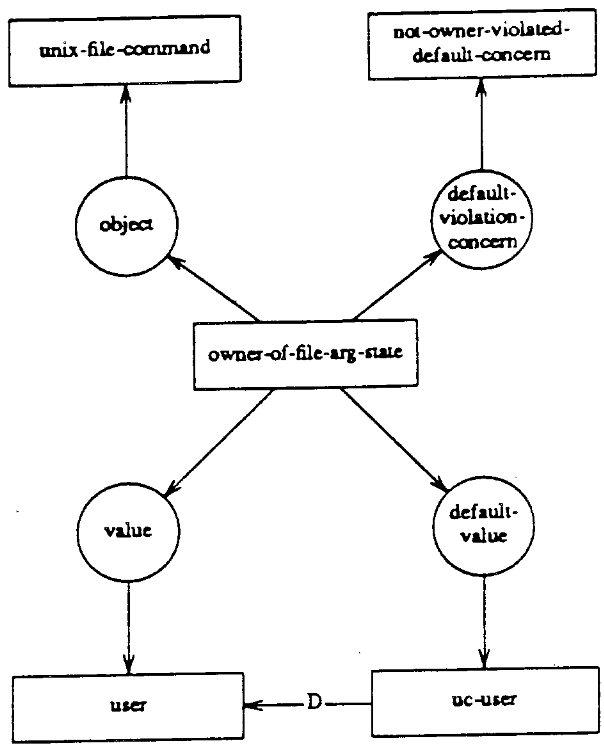

Figure 8.8: Representation of NOT-OWNER-VIOLATED-DEFAULT-CONCERN

concern called LS-SUN-MUST-BE-IN-DIRECTORY-CONCERN which is a child of NOT-OWNER-VIOLATED-DEFAULT-CONCERN. The representation of the violated default concern itself is similar to a default concern. Figure 8.9 represents that the LS-SUN-MUST-BE-IN-DIRECTORY-CONCERN as a concern of the LS-COMMAND-ON-SUN plan. The object of concern is the user who is the actor of the LS-COMMAND-ON-SUN plan. The condition of concern is the CURRENT-DIRECTORY-STATE of the same user. The desired value is that the current-directory be the same as the directory argument of the LS-COMMAND-ON-SUN plan.

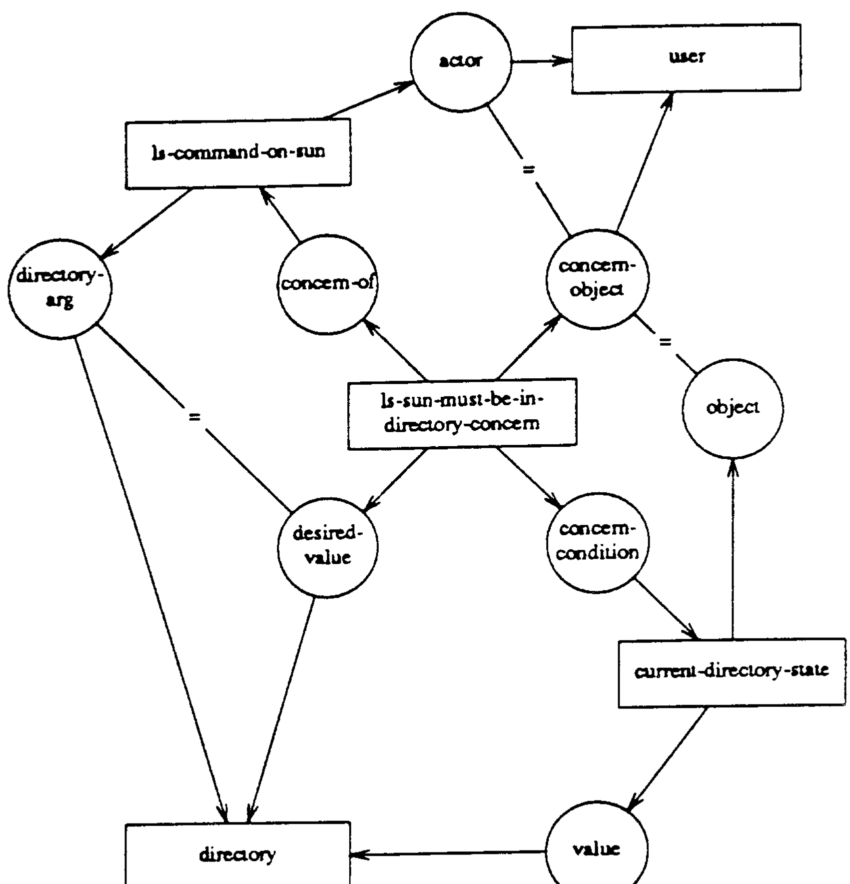

Figure 8.9: Representation of LS-SUN-MUST-BE-IN-DIRECTORY-CONCERN

Particular concerns have been entered into the database of UNIX plans through a KODIAK knowledge representation acquisition language called DEFABS. These concerns are all based on our experience using UNIX as well as discussions with other UNIX users in our research group. I am currently investigating a way to enter this concern information

using the UCTeacher program [22,24,23,38] a natural language knowledge acquisition system. Eventually, KIP may incorporate a learning component that will enable it to detect the frequency of certain plan failures and to store these as concerns.

8.7 Conclusion

Concerns allows a commonsense planner to detect plan failures among the many potential plan failures that might occur. Thus, KIP is able to determine whether its initial plans will be effective in particular planning situations.

This chapter has focused on *condition concerns*. KIP's use of condition concerns focuses KIP's attention on those conditions which are most likely to fail. KIP does not consider the many other conditions of a plan unless it has some information that a default has been violated. Therefore, concerns have addressed the major problem of CFD - focusing the attention of the planner on only those conditions which are relevant to plan failure.

I have discussed concerns in terms of the properties of algorithms that address the plan failure detection problem. Concerns address knowledge efficiency while allowing KIP to use default knowledge effectively. Concerns also seem cognitively valid. They are a way of representing and using knowledge that a consultant might have about the likelihood of particular condition failures. Specific/general concerns address the general knowledge application property by allowing KIP to deal with knowledge about condition failure at various levels of abstraction.

Much of this chapter has been about violated default concerns, designed to address the uncertainty property of commonsense planners. Violated default concerns are caused by default violations in unique planning situations. Violated-default/default concerns allow KIP to deal effectively with a large body of default knowledge. These type of concerns have proved to be the most interesting condition concerns, since they allow KIP to have different concerns in different situations.

The knowledge rich property of the commonsense planners is not discussed in this chapter. In order to adhere to this property, KIP must have a large amount of knowledge about the conditions which are necessary to execute plans successfully. In other words, if a plan failure due to condition failure will occur, KIP should have the knowledge to detect this condition failure. As discussed in the Chapter 1, there is an inherent tension between the knowledge-rich property and the knowledge-efficient property. As an algorithm is more knowledge-efficient, it is less knowledge rich. Rather than viewing concerns as a way of circumventing knowledge, concerns are viewed as a way of representing and using new knowledge that has not been previously represented. The knowledge regarding the likelihood that conditions will fail is just as important as knowledge about the conditions themselves.

In the following chapters, the usefulness of the concern idea is elaborated further.

Goal conflict concerns, which allow KIP to detect plan failures due to goal conflict, are discussed in Chapter 10. Goal conflict concerns are viewed as an even more powerful tool than condition concerns for focusing KIP's attention on potential plan failures. For a potential plan, there are many more potential plan failures due to potential goal conflict than due to condition failure.

Chapter 9

Detecting Plan Failures Due to Goal Conflict

9.1 Introduction

In Chapters 7 and 8, the means by which potential plan failures are detected is discussed. The detection process provides a planner with the means to determine if a potential plan is appropriate for a particular problem situation. There are two parts of plan failure detection: (1) condition failure detection and (2) goal conflict detection. The focus of the previous two chapters has been on condition failure detection. In the present chapter, I focus on how potential failures due to goal conflicts between an effect of a potential plan and a goal of the user are detected.

Detecting failures due to goal conflict is more complex than detecting failures due to condition failure. A potential plan might fail due to a limited number of conditions. In contrast, any of the effects of the same plan could potentially conflict with one of the many explicit or inferred goals of an agent. Since a commonsense planner is faced with a combinatorial explosion of potential goal conflicts, it cannot consider each potential goal conflict as a source of plan failure. Therefore, an algorithm is needed to limit those potential goal conflicts which should be considered.

In this chapter, I first give an example of a plan from UC in which KIP detects a goal conflict. This example reflects the difficulty inherent in the goal conflict detection problem. I then examine the properties of an algorithm that addresses this problem. Finally, I describe those issues that must be addressed by any commonsense planner (CSP) that detects goal conflicts. In the next chapter, an algorithm for detecting potential goal conflicts is described.

There are two ways in which a goal conflict can arise:

(1) a planner is given or infers two goals which conflict

(2) a planner selects a plan which conflicts with one of its goals

Thus, when given a set of goals, a CSP should detect the existence of any conflicts among any two of its goals. These include both goals the CSP is given and any goals it infers from the planning situation. Additionally, once a planner creates a potential plan for its goals, it should detect goal conflicts between the effects of the potential plan and any of its goals. Effects can conflict with other goals by causing states which are incompatible with these goals, or by making plans to achieve these goals impossible.

In this chapter, goal conflicts are discussed in the context of plan failure. Therefore, I focus on goal conflicts caused by conflicts between effects of selected plan and a planner's goals. However, many of the issues described in this chapter also apply to the detection of conflicts between goals without reference to a selected plan to achieve one of its goals.

9.2 An Example of Goal Conflict Detection

Suppose the user asks the following question:

(1) `How do I move a file named junk to a file named file1?`

KIP selects the USE-MV-COMMAND plan. KIP creates an individual instance of this plan for the particular problem situation of moving the file named junk, say the USE-MV-COMMAND1 plan. This plan is to execute the command mv junk file1. KIP needs to examine the USE-MV-COMMAND1 plan in order to detect any goal conflicts between an effect of the USE-MV-COMMAND1 plan and one or more of the goals of the user, both explicit and inferred.

One effect of this plan is that if file1 exists, it will be overwritten. Let us call this result the *destination file deletion effect*. This effect conflicts with the user's interest in having access to his file named file1, because once the USE-MV-COMMAND1 plan is executed, the user will no longer be able to access the file.

Detecting the goal conflict is difficult since KIP knows about a large number of potential effects of USE-MV-COMMAND1. For example, KIP might also consider the following effects of the USE-MV-COMMAND1 plan:

- If the file named junk does not exist, the following message is printed: mv: junk: Cannot access: No such file and directory

- The protection of the file named junk is the same as that of the file named file1.
- If the user does not have permission on the current directory, the following message is printed: mv: junk: rename: Permission denied.
- If the file named file1 is write protected, the user is asked: override protection 444 for file1?

In addition, KIP might consider the following effects that the USE-MV-COMMAND1 plan inherits from its parents in the hierarchy of plans:

- The directory inode will be updated.
- The disk arm will move due to a directory update.

KIP also needs to know about many interests of the user that could conflict with these effects. For example, KIP might also consider these user interests:

- Try to limit disk space usage
- Execute commands in a way that maintains a low load average
- Have a small number of files in each directory
- Keep your password secret

None of these user interests conflict with the effects of the USE-MV-COMMAND1 plan.

9.3 Properties of a Goal Conflict Detection Algorithm for a Commonsense Planner

The *goal conflict detection problem* (GCD) refers to the problem of detecting those goal conflicts that require goal conflict resolution. This problem occurs when a commonsense planner has created a potential plan for a goal of the user. A CSP should determine if any effect of the plan might cause a conflict with a goal of the user.

In Chapter 1, the properties of a CSP algorithm were introduced. In Chapter 7, I discussed three of these properties as they relate to the plan failure detection problem, with focus on the conflict failure detection problem. In this section, I discuss how these same properties relate to the GCD problem. Since goal conflict detection is more complex than

condition failure detection, additional constraints on each of these properties are necessary. The three properties discussed earlier are:

(1) *Knowledge Efficiency* - consider only those potential plan failures which are most likely to occur

(2) *Cognitive Validity* - attempt to model the method used by human consultants use to detect plan failures

(3) *Uncertainty* - ability to detect plan failures effectively when the planning situation is not fully specified

In addition, a fourth property is needed due to a CSP's dependence on interests during GCD:

(4) *General Knowledge Application* - ability to apply knowledge about general interests to specific situations

The properties of an algorithm that addresses GCD influence the criteria used to evaluate any algorithm for GCD. These criteria will be discussed in the following section.

Knowledge efficiency is even more important for goal conflict detection than condition failure detection due to the fact that both plans effects and user goals need to be considered. A weak method for solving the goal conflict detection problem would be to compare every potential effect of a particular plan with every explicit and inferred goal of the user. According to this method, if CSP knew about 5 effects of a plan and knew about 50 explicit and inferred goals of the user, CSP would need to make 250 tests for conflict. However, exhaustive search for potential goal conflicts violates the knowledge efficient property of commonsense planners. In order to avoid checking for conflicts between every effect of a plan and every potential goal of a user, we require some method of identifying which potential conflicts should be considered.

A human planner usually know right away that a certain plan will fail. According to the cognitive validity property, any algorithm from GCD should consider a small number of goal conflicts which a human planner might consider when trying to detect goal conflicts. This property reinforces the knowledge efficient property, i.e. consideration of a small number of potential goal conflicts.

The uncertainty propertyof CSP algorithms specifies that CSP should be able to rely on default situation knowledge in order to detect potential goal conflicts when only incomplete information is available. For example, CSP might have knowledge that unless

there is information to the contrary, it is likely that a file has read and write permission. If exhaustive search is used as an algorithm for GCD, a comparison of every effect with every goal might be difficult due to the CSP's dependence on default knowledge. Context-dependent defaults may entail a considerable amount of processing. Therefore, since much of the knowledge about a particular situation may be unknown, each individual comparison for conflict might entail much effort.

Default situation knowledge is not exact. CSP can only assume, based on the current planning situation, that various states are likely. Since CSP knows about the likelihood of various states, CSP can only determine the likelihood that a particular goal conflict will occur. Therefore, we assume that an algorithm for GCD will use this default situation knowledge to return a small number of the most *likely* potential goal conflicts in a particular planning situation.

The general knowledge application property is particularly important for GCD since many goals of the user are inferred by CSP, rather than being described by the user in a particular problem situation. These goal inferences are based on CSP's long-term knowledge about the user's long term interests in the general state of the world. An individual goal is often only inferred by CSP when there is some action that might threaten an interest of the user. This is another example of CSP applying its knowledge regarding general situations to particular planning problems. If CSP were to compare every effect of a plan with all the goals of the user, it would first need to examine each interest of the user. It would also need to determine if an individual goal should be inferred in the situation due to the particular interest. This process would violate the knowledge efficient property of commonsense planners. Very few of these interests will give rise to goals in a particular situation, and even fewer will become causes of goal conflict.

9.4 Criteria for Evaluating a GCD Algorithm

An algorithm that addresses the GCD problem should consider only a limited number of potential goal conflicts and ignore other potential goal conflicts completely. Such an algorithm should be judged according to its ability to detect the goal conflicts that will occur if a particular plan is used. It also should be judged according to how many potential goal conflicts the algorithm has to consider in order to detect these goal conflicts. One criteria for evaluating an algorithm is termed the *knowledge efficient criteria*. According to this knowledge efficient criteria, the size of the set of potential goal conflicts considered by an algorithm is compared with the size of the set of potential goal conflicts returned by an algorithm. The goal conflicts that an algorithm returns are those requiring goal conflict resolution.

A CSP algorithm which depends on default situation knowledge should return a small number of the most *likely* potential goal conflicts in a particular planning situation.

Thus, any part of a GCD algorithm can be evaluated according to the likelihood of a particular goal conflict that it returns. Let us call this criteria for evaluating a GCD algorithm the *likelihood criteria*. According to likelihood criteria, if an algorithm for GCD returns a potential goal conflict which is very unlikely, the algorithm should be modified accordingly.

The likelihood criteria is easier to evaluate than the knowledge efficient criteria. It is easier to assess the likelihood of specific potential goal conflict than the set of potential goal conflicts that a GCD algorithm might return. However, determining the likelihood of a potential goal conflict is a difficult process. For the purposes of this discussion, I assume that the likelihood criteria could be evaluated for a particular algorithm by a human expert with a great deal of UNIX experience. In principle, however, the information might be supplied by an analysis of data of actual UNIX interactions.

In this discussion, likelihood refers to two related but distinct concepts. Default situation knowledge describes the *likelihood* of a particular state in a particular planning situation. Knowledge about a particular state influences the *likelihood* of a conflict between an effect of a plan and a goal of the user. Detecting the most likely goal conflicts is the main task of any GCD algorithm.

Likelihood is also an important issue in selecting those interests which are applicable to a particular planning situation. Some general interests are more likely than others to give rise to corresponding goals in a particular planning situation. This likelihood can be determined by the importance of the interest to the user. For example, suppose that the user's interest in preserving the contents of his files is considered a more important interest than the user's interest in maintaining a low load average. Therefore, when both these interests might be threatened, it is more likely that goal inference will reflect the user's preserve-file interest than the user's maintain-low-load interest.

9.5 Taxonomy of Goal Conflicts

There are three types of goal conflict that should be detected by a CSP. These types of goal conflicts are defined by the effect of a selected plan resulting in:

(1) *Incompatible State* -	plan effect is incompatible with a user goal
(2) *Deleted Precondition* -	plan effect deletes a precondition necessary to satisfy a user goal
(3) *Consume Resource* -	plan effect consumes resource necessary to satisfy a user goal

This division, similar to Wilensky's three classes of negative goal relationships [35], differs from his formulation in two ways. First, his *Mutually Exclusive States* is divided into incompatible state and deleted precondition goal conflicts. This division distinguishes what Wilensky terms *plan conflicts*, conflicts that are due to the plans that are selected to satisfy user goals, from *proper goal conflicts*, conflicts between incompatible states. In my classification, the first category of goal conflicts refers to proper goal conflicts, while the second and third categories refer to plan conflicts. The distinction between *deleted precondition* goal conflicts and*consume resource* goal conflicts is less important than the distinction between proper goal conflicts and plan conflicts. However, as I describe in Chapter 10, the deleted-precondition/consume-resource distinction is useful during the goal conflict resolution process.

In addition, Wilensky's *Causing a Preservation Goal* category is deleted. According to my view, the detection of preservation or other background goals is pervasive in all three categories of goal conflict.

In the following section, the GCD problem is divided into three subproblems. I then discuss how a CSP should address each of these subproblems during goal conflict detection.

9.6 Dividing up the Goal Conflict Detection Problem

There are several issues that must be addressed by any algorithm that detects goal conflicts. I divide the GCD problem into the following three subproblems, based on the three types of objects that are considered during goal conflict detection: effects, interests, and potential goal conflicts. basic objects of goal conflict detection. (The objects that are considered in each of these subproblems are highlighted below.)

(1) *conflicting effect selection* - **effects**	find the effect of a plan that might conflict with a user goal
(2) *threatened goal selection* - **interests**	select the goal of the user that might conflict with an effect of a potential plan
(3) *goal conflict evaluation* - **potential goal conflicts**	evaluate the importance and seriousness of a potential goal conflict in the particular planning situation

Let us briefly examine these subproblems in the context of the example discussed earlier on page 127.

```
(1) How do I move a file named junk to a file
    named file1?
```

In this example, conflicting effect selection refers to the selection of the destination file deletion effect as an effect that is likely to conflict with a user goal. Threatened goal selection refers to the selection of the user's preserve-access interest as an interest that is likely to conflict with an effect of the potential plan. Goal conflict evaluation refers to the decision made regarding the importance of the file-deletion/preserve-access conflict in this particular situation. In this case, since CSP has no other information regarding the importance of the file named file1, KIP must rely on default knowledge regarding the user's interest in preserving his files. If it is unlikely that the user has a file named file1 and if CSP's default knowledge indicates that the user is only moderately interested in preserving his files, no goal conflict should be detected.

This division is not meant to reflect different parts of an algorithm that CSP might use. These subproblems are not necessarily ordered. CSP could, for example, select the goals which are likely to cause goal conflict before selecting the effects that are likely to cause goal conflict. However, goals are often only inferred as a result of being threatened by the effect of a particular plan. Therefore, it is convenient to consider these three subproblems in the order in which they are presented.

Likelihood of a goal conflict is the main criteria used by a GCD algorithm in deciding to view a potential goal conflict as one that needs to be resolved. Therefore, the factors that CSP considers in any GCD algorithm should determine the likelihood of a potential goal conflict. In addressing both conflicting effect selection and threatened goal selection, a particular effect or goal is *selected* for consideration due to its likelihood/unlikelihood as a participant in a potential goal conflict. In addressing goal conflict evaluation, a goal or goal conflict is evaluated according to its likelihood in a particular problem situation. Thus, each of the three subproblems of GCD involve factors that suggest the likelihood/unlikelihood of a potential goal conflict being considered in a particular problem situation.

9.6.1 Conflicting Effect Selection

The *conflicting effect selection* problem refers to the selection of those effects of a plan that are most likely to be considered as causes of goal conflict. Thus, the algorithm's task is to determine those effects of a particular plan which are most likely to create states which will cause a conflict with some other goal. This section thus focuses on the issues addressed by CSP in order to select the conflict-causing effects in different planning situations.

Conflicting effect selection differs from other aspects of the GCD problem in that the algorithm does not have difficulty determining concept relevancy. In any implementation of a planner, most of the effects of a particular plan should be both easily accessible to the planner and computable by the planner. These effects may be described as effects in the definition of a particular plan. Alternatively, these effects are inherited from plans which dominate the plan in the hierarchy. Rather than determining concept relevancy, the

major issue is to determine the likelihood that a particular effect will become a cause of goal conflict.

9.6.1.1 Incompatible States

Selecting a conflicting effect in an *incompatible state* goal conflict involves selecting the effects of a plan that are most likely to cause states incompatible with a user goal. In order to properly select such effects, a GCD algorithm must use likelihood knowledge. Such an algorithm should use knowledge regarding the likelihood that an effect of a particular plan is incompatible with a user goal, expressed or inferred. Those effects which have been a participant in many goal conflicts are considered likely causes of goal conflict.

For example, in (1), CSP should consider the destination deletion file effect of the USE-MV-COMMAND1 plan as a possible cause of goal conflict. However, the fact that file1 will have the same permission as temp, another effect of USE-MV-COMMAND1, should not be considered. This is true since our human expert claims that the effect most likely to cause goal conflict is the destination file deletion effect. The expert bases his likelihood assessment on his experience of goal conflicts which have occurred when executing this plan.

9.6.1.2 Deleted Preconditions

The *deleted precondition effect selection* refers to the selection of those effects of a plan likely to cause a state which will result in an unsatisfiable user goal. This unsatisfiable goal results when a precondition, necessary for a plan to satisfy the goal, is deleted by the selected plan. For example, suppose the user asks the following question,

(2) `How do I compact the file large, and then print it out?`

Suppose that CSP first selected the USE-COMPACT-COMMAND. CSP should consider the effects of this plan. After executing the plan, the contents of the file is stored in a special compacted form, which cannot be manipulated by some UNIX commands. In this example, the compaction effect results in the impossibility of file printout. There is no inherent conflict between printing a file named large, and the file being compacted. However, normal ascii format is a condition for any plan which could print out the file. Thus the effect of the USE-COMPACT1-COMMAND should be selected as a potential cause for goal conflict. Resolution of such goal conflicts may be accomplished by reordering plan steps. Deleted precondition goal conflicts is the type of goal conflict that NOAH [32] addressed by using a least commitment strategy.

In order to address the deleted precondition effect selection problem, knowledge is needed regarding the likelihood that an effect of a potential plan deletes preconditions

necessary for other user plans. This likelihood information is dependent on the particular planning situation. A GCD algorithm should consider only those user plans which are likely to be needed in order to satisfy user goals in the current planning situation. A GCD algorithm should select only those effects which are likely to delete preconditions of these other user plans.

9.6.1.3 Consume Resource Effect

The *consume resource effect selection* problem refers to the selection of those effects of a plan most likely to consume resources necessary to satisfy a user goal. For example, suppose the user asks the following question:

(3) `How do I run a lisp process in one window and a prolog process in another window on my Sun?`

Suppose that CSP creates the MOUSE-ON-LISP-MENU1 plan.When the user selects the lisp menu item with his mouse button, a lisp process is started. CSP should consider the consume resource effect i.e. this plan consumes a large amount of the memory resource. Therefore, it will likely become difficult to satisfy other user goals whose plans require memory as a resource. Once the memory resource is used up by a lisp process, it is impossible to run a prolog process simultaneously.

Therefore, in order to address the consume resource effect selection problem, a GCD algorithm must consider the likelihood that a plan will use resources that are required by other user plans in the particular problem situation. Like deleted precondition effect selection, a GCD algorithm should consider only those user plans which are likely to be needed in order to satisfy user goals in the current planning situation. A GCD algorithm should select only those effects which consume resources also needed by these other user plans.

9.6.2 Threatened Goal Selection

Threatened goal selection refers to the selection of the goals of the user that might conflict with an effect of a potential plan in a particular planning situation.

Threatened goal selection is more complex than conflicting effect selection for two reasons. Firstly, the threatened goal is not necessarily part of the description of a potential plan. Any of the many goals of the user are potential sources of goal conflict with the effect of a particular plan. Secondly, as described above, many of the goals that CSP needs to consider are not specified goals of the user. Instead, these goals are inferred in

response to an interest of the user being threatened by some state, e. g. the effect of a selected plan. Therefore, the problem of threatened goal selection is further subdivided into two problems:

(1) *threatened expressed goal selection*

(2) *threatened interest selection*

9.6.2.1 Threatened Expressed Goal Selection

The *threatened expressed goal selection problem* of GCD refers to selecting goals expressed by the user that may conflict with the effects of a potential plan. These expressed goals may be goals which the user has specified in his query. Alternatively, these expressed goals may be user goals which UC has inferred, but are not threatened interests. For example, suppose the user asks the following question:

(4) `I want to delete the file named boo, but I'd like to save the first 5 lines.`

CSP might first create the USE-RM-COMMAND2 plan. One effect of this plan is that the entire contents of the file named jim would be deleted, including the first 5 lines. CSP should select the user's expressed goal of keeping the contents of the first five lines of the file named boo as a potential source of goal conflict. This should be done even though CSP might not assume that users generally want to preserve a small part of their files.

Therefore, a GCD algorithm must have knowledge regarding the likelihood that an expressed goal will take part in a goal conflict. The user may have chosen specific goals that conflict with specific effects. These effects may have been determined by the knowledge-base designer as unlikely sources of goal conflict. A GCD algorithm must select the expressed goals that are most likely to conflict with an effect of a potential plan.

9.6.2.2 Threatened Interest Selection

The *threatened interest selection problem* of GCD refers to the selection of the potential interest with which a particular effect of a plan may conflict. Since there are many interests that CSP assumes on the part of the user, CSP should choose the particular interest(s) with which a particular effect might conflict.

For example, suppose the user asks the following question:

(5) `How do I copy a file named paul to the file named lisa?`

CSP might select the USE-CP-COMMAND plan and create the USE-CP-COMMAND1 plan for this particular situation. One of the effects of USE-CP-COMMAND1 is that if the file named lisa exists, it will be deleted. In order to address the threatened interest selection problem in this situation, CSP should select the user's general interest of preserving access to his files as a potential candidate for conflict with an effect of the USE-CP-COMMAND1 plan. The *preserving access interest* should be selected. It is likely that this general interest will be threatened if the USE-CP-COMMAND1 plan is executed. In contrast, CSP should not select the user's general interest in having a low load average. It is unlikely that executing the USE-CP-COMMAND1 plan will threaten this goal or interest. CSP needs some means of selecting the preserving access interest among the many interests of the user.

Unlike threatened expressed goal selection, the interest which is selected must be evaluated in the particular planning situation. This aspect of threatened interest selection is termed *threatened interest evaluation*. Although CSP knows that the user's certain general interest might be threatened by a potential plan, CSP does not know if the general interest is applicable in this particular situation. If the interest of the user is not really threatened in this particular problem situation, then the interest should not be considered further. However, if the interest is threatened, an individual goal should be inferred that reflects the interest in this particular situation. The interest is evaluated by examining relevant knowledge about this particular planning situation in the knowledge base.

For example, in (5), although CSP might assume that the user has a general interest in preserving the contents of his files, usually CSP will not have more specific information. For example, CSP will generally not know that the user has an individual goal of preserving a particular file named lisa that belongs to him. If CSP knows that a file named lisa already exists, then CSP might create a goal of preserving lisa. Therefore, a goal conflict might be created between the individual goal of preserving the file named lisa and the deletion effect of the USE-CP-COMMAND1 plan. If it is unlikely that lisa exists, then it is unlikely that the user has such a goal of preserving a file named lisa. In that case, an individual goal of preserving lisa may or may not be inferred. If CSP has particular information that no file named lisa exists, no individual goal of preserving such a file should be inferred.

Therefore, depending of the information a CSP has about the particular interest, a CSP make decisions about the likelihood that such a goal exists. If a CSP decides that such a goal is unlikely, no goal should be instantiated. Otherwise, a goal should be created that reflects the interest and also has contains information about the importance of that goal in the current planning situation.

Threatened Interest Evaluation with respect to the Query Goal

A special case of threatened interest evaluation is the evaluation of the threatened interest with respect to the goal expressed by the user in his query. The threatened interest has been selected as an interest likely to be threatened by the effects of a potential

plan. CSP must determine if the threatened interest actually conflicts with the effect of the potential plan, considering the goal expressed in the user's query. In order to address this problem, CSP should examine the relationship between the user-expressed goal contained in the query for which CSP is trying to find a plan and the selected threatened interest of the user. For example, suppose the user asks the following question:

(6) `How do I delete the file file3?`

CSP might select the USE-RM-COMMAND plan and create the USE-RM-COMMAND1 plan for this particular situation. Suppose that CSP had selected the conflicting effect of the USE-RM-COMMAND1 plan that file3 is deleted. CSP might also infer a threatened goal of preserving the file file3. CSP might infer the goal in response to the threat against the long term interest of the user in preserving his files. However, the user has specified in his query that his goal is to delete the file named file3. The apparent conflict between deleting file3 and preserving the contents of file3 is an *artificial* conflict. CSP should assume that when the user specifies a goal not generally in his best interest, he is aware that the goal is not in his best interest. However, the user desires the goal regardless. CSP should assume that since the user has specified the goal of deleting the file named file3 in his query, he really does have the goal of deleting this file. Therefore, the goal of preserving file3 should not be inferred.

However, if the user asks the following question:

(7) `I want to free up disk space and file4 is using up a lot of disk space. What should I do?`

CSP might also choose to select the USE-RM-COMMAND plan as a potential plan in this query, since the USE-RM-COMMAND is stored as a plan for freeing up disk space. Since the user has mentioned that file4 uses up a lot of disk space, file4 would seem like a good choice for removal, using the individual plan USE-RM-COMMAND2. The user has not expressed his desire to delete file4. Therefore, it is reasonable to assert a *real* goal conflict between the conflicting effect of the USE-RM-COMMAND2 plan of deleting the file file4, and the individual goal, inferred by CSP, of preserving the contents of file4. The assertion of a real goal conflict appears reasonable because deleting the file is only a side effect of the USE-RM-COMMAND2 plan. This effect was not part of the user's intention as expressed in his query. It is a side effect of a plan selected to satisfy the goals of the user. Since file deletion is not the intended effect of the user, the goal of preserving file3 should be inferred. In (6), however, since deleting the file is the intended effect of the USE-RM-COMMAND1 plan, the file preservation goal should not be inferred.

In (5), the relationship between the effect of the USE-CP-COMMAND1 plan of deleting the file lisa, and the user-expressed goal of moving the file paul is more complex. Since

the user has expressed that he wants to move the file paul to the file lisa, it might seem that he really doesn't want the file lisa to remain in its original form. On the other hand, the effect of deleting the file lisa might be interpreted as a side effect which is not a goal of the user. Thus, this file deletion effect might not be a user goal. Evaluation of the threatened interest with respect to the query goal during the goal conflict detection phase is essential. This evaluation enables an algorithm for CSP to resolve only *real* goal conflicts and to disregard *artificial* goal conflicts before goal conflict resolution.

An algorithm for CSP could evaluate the threatened interest with respect to the query goal in a number of different ways. The algorithm could first infer a threatened goal that reflects the interest and later realize that this goal should be disregarded. Alternatively, an algorithm could choose not to select the threatened interest at all. However, as shown in this section, the threatened interest must be evaluated with respect to the query goal at some point in any algorithm for CSP.

The evaluation of threatened interests with respect to the query goal might have been categorized as a kind of goal conflict evaluation instead of a kind of threatened interest evaluation. However, the present classification was chosen to reflect the fact that evaluations of threatened interests with respect to the query goal are possible without referring to the conflicting effect. Such evaluations are therefore possible in other planning situations where no potential plan has been selected, but an interest has been selected and must be evaluated. In the following section, I discuss goal conflict evaluation. Goal conflict evaluation assumes that both the conflicting effect and the threatened interest have already been selected and evaluated.

9.6.3 Goal Conflict Evaluation

Goal conflict evaluation refers to the decisions made regarding the importance of a potential goal conflict in a particular planning situation. The GCD algorithm's task is to assert goal conflicts which must be resolved. This section describes the way in which CSP should decide whether a goal conflict is important enough for goal conflict resolution. If CSP determines that a goal conflict meets its criteria for importance, the goal conflict should be asserted and passed to the goal conflict resolution mechanism.

In order to determine the importance of a potential goal conflict, CSP should evaluate both the likelihood of the goal conflict and the seriousness of the goal conflict. This evaluation is done by analyzing the likelihood and importance to the user of the *conflicting effect* of the plan and the *threatened goal* which is threatened by this effect. Therefore, any algorithm that addresses the goal conflict evaluation problem of GCD should have some mechanism for weighing the following four factors:

1. the importance of the user goal for which the potential plan has been created

2. the likelihood of the conflicting effect's occurrence when the plan is executed in this particular problem situation

3. the importance of the goal which might be threatened by the conflicting effect

4. the seriousness of the threat to the threatened goal, i.e. if the conflicting effect occurs, the likelihood that the goal will be threatened

For example, suppose the user asks the following question:

(8) `I like Michael's .login file better than mine. How do I use his file instead of my file when I log in?`

CSP might create the USE-CP-COMMAND2 plan to copy Michael's .login file to the user's home directory, by typing cp ~michael/.login ~/.login. CSP should know that the destination file of USE-CP-COMMAND2, the user's own .login file, exists. Therefore, if the user executes the USE-CP-COMMAND2 plan, the contents of the user's file named .login will almost certainly be lost. Thus, the likelihood that the conflicting effect will occur is very great. CSP should assume that the user's goal of using Michael's .login at login time is important, since he has stated this goal in his query. However, the usual intended goal of the USE-CP-COMMAND2 plan, i.e. having a copy of Michael's .login in the user's home directory, is not a goal of the user.

CSP should detect the user's threatened goal of preserving access to his own .login because users generally have an interest in preserving their files. Since the interest should be stored as an important interest of the user, the goal should be considered an important one as well. Since the likelihood of the conflicting effect is high, and the seriousness of violating the threatened goal is also high, CSP should consider this an important goal conflict. Therefore, CSP should assert a *real* goal conflict might occur in this particular problem situation. CSP should pass this goal conflict to the goal conflict resolution mechanism.

At times, although the conflicting effect may be very unlikely, a goal conflict should still be detected. For example, suppose the user asks the following question:

(9) `How do I print out my midterm on the imagen printer?`

Use the office printer, using ditroff -Poff, so that others don't read it.

In (9), CSP should use its knowledge about midterms to detect a serious, though unlikely, goal conflict. CSP might create the USE-DITROFF-COMMAND1 plan to print out the midterm on the imagen. Suppose that CSP selects the conflicting effect of this plan that

others might read the midterm output of the midterm in the printer room. (The imagen is in a public place). This conflicting effect might be unlikely. Few people are in the printer room at any one time, and most people do not read the output of other user jobs coming out of the printer. However, suppose that CSP would select the threatened goal of the user keeping the midterm secret. CSP should assume that this is a very serious threat to an important user goal. Although the conflicting effect is relatively unlikely, a goal conflict should be instantiated due to the seriousness of the threat to the user's goal.

Example (5) is more difficult due to the lack of information about the particular planning situation. If no other information is given, CSP might decide that it is only somewhat likely that the destination file named lisa exists. Therefore, the conflicting effect of the file named lisa being deleted is not very likely. But even if the file named lisa did exist, the importance of the goal of deleting this file is questionable, since it is unclear if the user wants the contents of the destination file or not. Therefore, a goal conflict may or may not be instantiated.

The rules for goal conflict evaluation in the extreme cases are simple. For example, if the user goal is important, and conflicting effect is likely, the threatened goal is important, and the threat is serious, an algorithm for CSP should instantiate a goal conflict. However, in cases where one or more of these factors is mediocre in importance or likelihood, it is less clear whether a goal conflict should be instantiated.

The goal conflict evaluation problem interacts with the other problems of GCD discussed above. Due to this interaction, any algorithm for GCD cannot address each of the problems separately. For example, in (9), suppose that CSP created the USE-DITROFF-COMMAND1 for the goal of printing out the midterm. If CSP were to consider the problems in the order in which they were described, CSP would first search for the most likely conflicting effects of the USE-DITROFF-COMMAND1. CSP might discover the very unlikely conflicting effect of others reading the output during the conflicting effect selection stage. Since the conflicting effect is unlikely, this effect might be disregarded. Thus, CSP would not detect the goal conflict between this effect and the serious threat it poses to the user's goal of keeping the midterm secret. Therefore, a knowledge efficient algorithm that addresses GCD should be able to deal with these problems in a unified manner. Otherwise, a knowledge-efficient algorithm might be forced to consider each potential goal conflict between every effect of a plan and every goal or interest of the user. Such an algorithm is presented in the following chapter.

9.7 Summary of Goal Conflict Detection

Detecting plan failures due to goal conflict is a difficult problem for a commonsense planner. The difficulty is mainly due to the large number of goals with which a plan could potentially conflict. An algorithm for goal conflict detection might need to exam-

ine a combinatorially large number of potential goal conflicts. Therefore, a commonsense algorithm for GCD, which uses its knowledge effectively, must be devised.

In this chapter I have described some of the issues that a GCD algorithm must address. I have described these issues in terms of three subproblems of GCD: (1) conflicting effect selection, (2) threatened goal selection, and (3) goal conflict evaluation. In the following chapter, I describe a potential commonsense algorithm for GCD, based on the concept of concerns described in Chapter 8. Two aspects of the algorithm must be evaluated. First, I must assess the algorithm's ability to address the GCD subproblems described in this chapter. Secondly, I must assess the degree to which the GCD algorithm adheres to the GCD algorithm properties described above.

Chapter 10

Goal Conflict Concerns

10.1 Introduction

In the previous chapter, I described the difficulty and importance of the Goal Conflict Detection problem. This problem was described in terms of CSP, our generic commonsense planner. In the present chapter, I describe an algorithm that addresses the GCD problem. I describe this algorithm in terms of KIP, since this is an algorithm used by our commonsense planner for the UNIX operating system. The algorithm described is used by KIP to detect goal conflicts between an effect of a potential plan proposed by KIP and some background or expressed goal of the user.

Chapter 8 introduced the idea of a *concern*, used to detect potential plan failures. Chapter 8 focuses primarily on condition concerns, while this chapter, focuses on goal conflict concerns. In this section, I compare concerns about goal conflicts with concerns about condition failure. Condition concerns, discussed earlier, refer to those aspects of a plan that are likely to cause plan failure due to a condition of the plan that is needed for successful execution. *Goal conflict concerns* refer to those aspects of a plan which are likely to cause plan failure due to a potential goal conflict between an effect of a plan and a goal of the user. Goal conflict concerns are represented as three way relations between the selected plan, the conflicting effect, and the threatened goal.

The conditions about which KIP is concerned are always conditions of a particular plan. Goal conflict concerns, however, relate plans to user goals and to other pieces of knowledge that are not part of the plan. Examples of this knowledge include user interests which may or may not be threatened by the plan. Since these interests are usually not considered until such a threat is perceived, goal conflict concerns often refer to conflicts between a potential plan and an interest of the user. *Stored goal conflict concerns* refer to concerns about conflicts of interest. These are concerns about the selected plan conflicting with an interest of the user. If KIP detects a conflict-of-interest concern, then KIP must determine if it should infer an individual goal on the part of the user that reflects this interest.

If KIP decides to infer this individual goal, then a *dynamic concern* between the selected plan and the individual goal is also instantiated.

KIP's basic algorithm for dealing with goal conflict concerns is similar to the three-stage process described for condition concerns. First, KIP gathers the concerns by looking at the potential plan and other information from the planning environment. Second, KIP evaluates these concerns in the present planning situation. Thirdly, KIP decides how these concerns will influence the rest of the planning process. However, an algorithm for dealing with goal conflict concerns is more complex due to the need to evaluate both the effects of the plan and user interests and goals in the particular planning situation. In addition, due to the greater complexity of the GCD problem, more types of concerns need to be introduced. These other types of concerns are introduced in section 10.4 on page 147.

10.2 An Example of the Use of Goal Conflict Concerns

Before describing the other types of goal conflict concerns, and KIP's complete algorithm for dealing with these concerns, we consider a simple example of the use of goal conflict concerns. Suppose the user asks the following question:

(1) `How do I change my password?`

KIP is passed the goal of changing the password of the user. KIP's knowledge base contains a stored plan for the goal of changing a password, namely, the USE-PASSWD-COMMAND plan. As the effect of the USE-PASSWD-COMMAND plan, the user's password is changed on his current machine. This effect conflicts with the user's interest of having the same password on all machines on which he has an account. Let us call this interest the *identical password interest*. This user interest is a subgoal of the user being able to remember his password. The stored goal conflict concern which KIP has identified is a three-way relation between the USE-PASSWD-COMMAND, the effect that the password is changed on this machine, and the identical password interest.

Thus, in the concern identification step, KIP has addressed both the *conflicting effect selection* and *threatened interest selection* problems of GCD. KIP addresses these problems by storing a concern between the effect that is likely to cause goal conflict and the interest with which it is likely to cause goal conflict.

KIP next evaluates the identical password interest in this particular planning situation. KIP uses knowledge about the particular user to determine if an individual goal should be inferred in this situation which reflects the identical password interest. For example, if KIP knows specifically that the user has an account only on the current machine, then KIP does not assert an identical password goal for the user. Consideration of this goal conflict concern stops. KIP may have information that the user has accounts on many machines, or may make this assumption based on default knowledge from the user model. In

either of these cases, KIP then asserts an individual identical password goal on the part of the user. This step addresses the *threatened interest evaluation* problem of GCD.

KIP next evaluates whether the goal of the user is the intended effect of the selected plan. According to KIP's input, the user has specified that his goal is to change his password. KIP's knowledge base stores the fact that the USE-PASSWD-COMMAND plan is the best plan to accomplish this password-change goal. Thus, KIP determines that the goal of the user is the intended effect of the plan. KIP assumes that the concerns indexed under the USE-PASSWD-COMMAND plan are not concerns about the intended goal of the plan. Therefore, KIP assumes that the conflict between the password-changing on the current machine and the identical password interest is a real goal conflict, not an artificial goal conflict. The step addresses *query goal evaluation* problem of GCD.

KIP next evaluates this potential goal conflict. In this case, the concern about multiple passwords is marked as having a high degree of concern, and therefore a goal conflict is inferred. This potential goal conflict is then passed to the goal conflict resolution mechanism. This step addresses the *goal conflict evaluation* problem of GCD.

10.3 Properties of Concerns

Goal conflict concerns allow KIP to adhere to properties of a GCD algorithm described in the previous chapter. Like condition concerns, goal conflict concerns are primarily directed at the knowledge efficient property of GCD algorithms. Satisfaction of this property is even more important for detecting goal conflicts due to the large number of potential goal conflicts that algorithm might need to consider. KIP need not consider the many effects of a particular plan for potential goal conflicts. Also, KIP need not evaluate many user interests not known to be threatened a particular plan.

In the remainder of this section, I describe how two properties of concerns relate to goal conflict concerns:

(1) - degree of concern

(2) - specificity of concern

I describe how these properties of goal conflict concerns differ from the properties of condition concerns. I also present complex KIP examples which use these properties of goal conflict concerns.

10.3.1 Degree of Concern

Some goal conflicts are more likely to occur than other goal conflicts, and some goal conflicts are more important than others if they do occur. The representation of goal

conflict concerns reflects this difference by assigning a varying degree of concern to the stored concerns in the knowledge base. There are many factors that determine the degree of concern about a conflict-of-interest. The planning knowledge base designer needs to determine how likely a conflicting effect is to occur, how likely it is that the user holds the threatened goal, and how important this goal is to the user. The likelihood and importance information is needed by KIP to address the *goal conflict evaluation* problem of GCD. Instead of determining all this information every time a particular plan is selected, KIP encodes all this information as a degree of concern. For example, suppose the user asks the following question:

(2) `How do I print out a file on my terminal?`

Suppose KIP has chosen the USE-CAT-COMMAND plan. KIP has two stored concerns about this plan. KIP's first concern is a concern regarding the effect of the USE-CAT-COMMAND plan that a long file will continue to scroll on the terminal until the end of the file is reached. KIP's concern is that this scrolling effect conflicts with the user's interest of examining the contents of the file. KIP has not been given any other information about the file or the user's goals. Thus, KIP should assume that it is somewhat likely that the file is longer than the number of lines on the terminal. It is very likely that the user has the goal of wanting to see the contents of the file. If the user has this goal, then the goal is important to him. All this information can be encoded by KIP as a moderately high degree of concern.

KIP's second concern about the USE-CAT-COMMAND is the concern that if the file contains control characters, the terminal will be put in an unusable state. These control characters may be interpreted by the terminal as cursor commands and cause the terminal to lock up, erase information on the screen, or print nonsense on the screen. This screen locking effect conflicts with the user's interest in having a usable terminal. Since KIP has not been given additional information, it should assume that this screen locking effect is very unlikely. It is unlikely that the user's file contains control characters. Even if the file has control characters, the screen may not lock up. However, the threatened goal of having a usable terminal is assumed to be an important goal of the user. It is sometimes very difficult for the user to return the terminal to a usable state. A naive user might not know how to reset his terminal at all and would not be able to continue to work at all. In this case, by weighing the likelihood of the effect and the importance of the goal, this concern is assigned a low level of concern. Therefore, a goal conflict is not instantiated.

The degree of a concern is not an absolute, but a relative value. For example, suppose that the user has informed KIP that he is worried about a bad side effect of a particular plan. KIP should consider all possible concerns that it knows about. Even concerns marked as having a low degree are considered fully. KIP must therefore decide on a threshold value for the degree of concern in a particular planning situation. Concerns which have a degree of concern higher than that threshold are considered. Concerns which have a degree of concern lower than that threshold are ignored.

10.3.2 Specificity of Concern

Goal conflict concerns are hierarchical like the other objects in KIP's knowledge base. Stored concerns about conflicts of interest are therefore inherited from plans further up in the hierarchy. KIP exploits this property by assigning general concerns about goal conflict regarding a class of plans at a high level in the hierarchy. In this way, these concerns do not need to be defined for each of the children of these general plans. In addition, very specific concerns about goal conflicts can be associated with plans for specific planning situations. Since the degree of a goal conflict concern depends on many factors, KIP's knowledge base often includes specific goal conflict concerns which differ from their parents only in the degree of concern. For example, suppose the user asks the following question:

(3) `How do I print out an executable file on my terminal?`

KIP has specific information about printing such files on a terminal. KIP concretes the user's goal of printing files to the specific goal of printing executable files on a terminal. As in example (2), the USE-CAT-COMMAND is a plan for this goal. The structure of this plan is exactly the same as in example (2), except that the goal conflict concern regarding the screen locking of the user's terminal is of a very high degree. This concern has a high degree because of the high likelihood of the conflicting effect of putting the terminal in an unusable state. (The seriousness of the threat to the user's interest is the same.) KIP has no additional information in this particular planning situation. Therefore, during the evaluation of the screen locking concern, KIP asserts that the USE-CAT-COMMAND has a high degree of concern about locking the user's screen. KIP will therefore try select another plan that does not have this potential plan failure.

10.4 A Taxonomy of Goal Conflict Concerns

In order to address the GCD problem, a number of different dimensions of goal conflict concerns are necessary. These concerns do not address one of the three problems of GCD: conflicting effect selection, threatened goal selection, and goal conflict evaluation. Instead, KIP uses a unified approach and tries to address all of these problems simultaneously. These different dimensions of concerns are meant to address goal conflict detection in unique situations. Goal conflicts are most difficult to detect when planning in unique situations. However, in our experience plan failures due to goal conflict are most likely to occur precisely in unique situations. In the following section, I describe KIP's goal conflict concern algorithm. I describe how these dimensions of concerns are used together in the context of the algorithm.

The eight dimensions of concern discussed in this section are briefly described below, with reference to the type of conflict to which each concern refers. These concerns are grouped in pairs according to the criteria which distinguishes them from other concerns. (The criteria are highlighted in boldface.) Since these criteria are orthogonal, there may be some overlap between these goal conflict concern dimensions. The concerns are discussed in pairs, according to these criteria in the remainder of this section.

Source of Concern

Conflict-of-interest Concerns -	KIP-detected concern regarding a conflict between an effect of a plan and a user interest
Expressed Goal Conflict Concerns -	concerns expressed by the user between an effect of a plan and a goal of the user

Intended Effect of Potential Plan

Intended Effects Concerns -	concerns regarding a plan which has been used for its intended effect
Unintended Effects Concerns -	concerns regarding a plan which has been used for an effect for which it was not intended

Default Situation Knowledge

Default Concerns -	concerns in normal planning situations
Violated-Default Concerns -	concerns in situations where some defaults have been violated

Plan/Effect Distinction

Plan Goal Conflict Concerns -	concerns about conflicts caused by execution of a particular plan
Effect Goal Conflict Concerns -	concerns about a particular effect without reference to a particular plan

10.4.1 Conflict-of-interest Concerns vs. Expressed Goal Conflict Concerns

As I described earlier, stored goal conflict concerns are concerns about conflicts-of-interest. This is because KIP's long term knowledge about plan failure due to goal conflict describes general problems that are to be avoided. Dynamic concerns are those concerns that KIP creates during the planning process. Dynamic concerns are created in response to effects of a selected plan which threaten an interest of the user. However, there are some dynamic concerns that KIP does not create. These dynamic concerns are passed to KIP as part of its inputs. For example, suppose the user asks the following question:

(4) `How do I copy the file named init.lisp to the file aux.lisp without losing the contents of aux.lisp?`

In this case, the user has expressed an explicit concern regarding the effects of a potential plan that the user has in mind - probably the USE-CP-COMMAND. Namely, a particular goal, preserving the contents of file name aux.lisp, might be threatened by a potential plan for copying the contents of init.lisp to aux.lisp. In (4), the user has stated a expressed goal conflict concern. KIP knows about this goal conflict concern and it is viewed as an important potential goal conflict.

Of course, the user could have expressed a concern that was unwarranted. For example, suppose the user asks the following question:

(5) `How do I copy the file named init.lisp to the file aux.lisp without losing the contents of init.lisp?`

KIP knows of no way in which the contents of the file init.lisp might be lost while trying to copy this file. However, this is still an expressed goal conflict concern with which KIP must deal.

Since the expressed goal conflict concern in (4) is warranted, and the expressed goal conflict concern in (5) is unwarranted, KIP treats these concerns differently. In (4), KIP selects the USE-CP-COMMAND and determines the concerns for this selected plan. It evaluates each of these concerns, as shown above. Due to the user's expressed goal conflict concern, KIP adds some additional steps. Thus, KIP first identifies the potential goal conflict between the conflicting effect of the file aux.lisp being deleted and the interest of preserving the contents of the file aux.lisp. KIP then compares that potential goal conflict against the user's expressed goal conflict concern. Since the user has expressed a concern about a known potential goal conflict, KIP tries to resolve the conflict.

Example (5) differs from (4) in that KIP is unable to match any of its identified potential goal conflicts against the expressed goal conflict concerns expressed by the user. None of these potential goal conflicts threaten the goal of preserving the contents of the file init.lisp. Therefore, KIP examines all of the effects of the selected plan to determine if any of the effects of USE-CP-COMMAND conflict with the goal of preserving the contents of the file init.lisp. Since none of these effects conflict with the goal, no goal conflict is instantiated. The fact that the user's expressed goal conflict concern was unwarranted is reported in UC's answer to the user.

Conflict-of-interest concerns are checked before checking the explicit concerns of the user. When a user mentions an explicit goal, he is usually referring to a concern regarding potential plans that might conflict with this goal. Expressed goal conflict concerns are usually instances of conflict-of-interest concerns about which KIP is aware. KIP evaluates all of the potential concerns about conflicts-of-interest of a selected plan before looking at the expressed concerns of the user. By doing this, KIP can benefit from its stored information regarding these concerns in long term memory. The process is analogous to a consultant, hearing about a problem with a plan, who first compares that problem to problems that he has previously encountered.

10.4.2 Intended Effects Concerns vs Unintended Effects Concerns

When KIP is unable to find a stored plan that solves the goals of the user, it uses another plan that is used for some other similar goal. For example, suppose the user asks the following question:

(6) `How can I free up disk space?`

If KIP has no stored plan for this goal, it might select the USE-RM-COMMAND plan to accomplish the goal of the user. As an effect of that plan, disk space of the file that is removed is marked as free. One of the problems with using this plan is that it conflicts with the user's interest in preserving his files. The conflicting effect of using the USE-RM-COMMAND plan on a particular file is that the file is removed. This effect conflicts with user's goal of preserving the individual file. Let us call this conflict the *preservation/removal* conflict. In order to detect this goal conflict, a concern should be accessed between the removal of the file and the preservation of the file. The preservation/removal goal conflict should be passed to the goal conflict resolution mechanism.

However, suppose the user asks the following question:

(7) `How can I remove a file?`

In example (7), KIP would also select the USE-RM-COMMAND plan in order to satisfy the user goal. The USE-RM-COMMAND plan is defined as a plan in service of the

goal of removing a file. In this example, however, the user actually intends to remove a file. Therefore, the goal of preserving this file should not be threatened and the preservation/removal conflict should not be detected. Therefore, in example (7), KIP should not access a concern about the conflict-of-interest between removing a file and preserving a file.

As discussed in Chapter 9, this type of goal conflict between an intended effect and a general interest is termed an *artificial* goal conflict. Artificial goal conflicts should not be passed to the goal conflict resolution mechanism. This problem often occurs when a planner does not properly evaluate the threatened interest with respect to the query goal.

In order to avoid detection of artificial goal conflicts, KIP differentiates between *intended effect concerns* and *unintended effect concerns*. Intended effect concerns refer to conflicts-of-interest in which the selected plan is being used for its usually intended effect. Unintended effect concerns refer to conflicts-of-interest in which the selected plan is being used for some other effect of the plan. KIP always accesses a plan's intended effect concerns. A plan's unintended effect concerns are only accessed when the user goal is not the intended effect of the plan.

In example (7), since the USE-RM-COMMAND is being used for its intended effect, the concern about file deletion is not even considered. In example (6), however the USE-RM-COMMAND plan is being used for an effect other than deleting a file. Therefore, all unintended effect concerns are considered, including the concern that using the USE-RM-COMMAND plan will conflict with the user's goal of preserving his files.

In KIP, if a particular action is used as a plan for more than one goal, two different plans are created. For example, once KIP knows that a plan for freeing up disk space is to use the rm command, it creates a USE-RM-COMMAND-FOR-DISK-SPACE plan. This plan has a different intended effect than the USE-RM-COMMAND-FOR-DELETING plan. Therefore, this plan will have different intended effect concerns and unintended effect concerns than the USE-RM-COMMAND-FOR-DELETING plan.

Therefore, when KIP has selected a plan for the goal which it has been intended, KIP must check all the intended effect concerns for that particular plan. If KIP has selected a plan in order to satisfy a goal for which the plan has not been intended, then both the intended effect concerns and the unintended effect concerns must be checked.

10.4.3 Default Concerns vs. Violated-Default Concerns

The previous concerns have all been *default goal conflict concerns*. Default goal conflict concerns are those concerns with which the planner is usually occupied, given a set of defaults used to make assumptions about the world. When these defaults are violated, new concerns arise which I call *violated-default goal conflict concerns*. For example, sup-

pose the user asks the following questions:

(8) `How do I print out a file on the lineprinter?`

(9) `How do I print out a very long file on the lineprinter?`

In example (8), KIP selects the USE-LPR-COMMAND plan. Since no defaults have been violated, only default concerns are accessed. KIP finds no goal conflict concerns for this plan, and it is returned as a plan for accomplishing the user's goal.

In example (9), KIP also selects the USE-LPR-COMMAND plan. However, in this example, KIP should also access the violated-default concerns of the plan. Since the size of the default file is usually assumed to be short, the fact that the user has specified that the file is a long one violates a default. Because this default is violated, KIP accesses the violated-default concerns for the plan that reflect this violated default. KIP accesses a conflict between the effect of printing out a long file and the interest of being considerate to other users of system resources.

KIP always accesses the default goal conflict concerns of a particular plan. A plan's violated-default concerns are only accessed when a default is violated. However, only those violated-default concerns that reflect the violated default are accessed.

Accessing only the concerns that reflect the violated default makes the implementation of violated-default concerns difficult. In example (9), for instance, the violated-default concern is detected by accessing a category of plans that manipulate long files. Let us call this category the *long-file-plans* category. Since the default of the file being short is violated, a USE-LPR-COMMAND1 plan is created. The USE-LPR-COMMAND1 plan is dominated by both the USE-LPR-COMMAND plan and the category of long-file-plans. The individual plan inherits all the concerns of both parents. USE-LPR-COMMAND1 thus inherits conflict-of-interest concerns from both the USE-LPR-COMMAND plan and the long-file-plans category. KIP knows that concerns arising from the long-file-plans category are violated-default concerns. The relationship between long-file-plans category and USE-LPR-COMMAND1 is created in order to reflect the violated default.

One advantage of this implementation is that general concerns about non-default situations can be stored in one general category. For example, The long-file-plans category and its concerns are also used by KIP to detect other similar conflicts. KIP can use this category to detect conflicts regarding the use of system resources when compiling a very large program, typesetting a long paper, or sending a very large file over the network.

There can also be more exacting descriptions of violated-default concerns in a more specific category. For example, sending a 10 page file to the lineprinter might not be considered excessive and therefore would not generate a concern. However, sending

the same size file to the laser printer, which takes much longer to print, would generate a concern. Therefore, a specific concern category of plans which print large files on a laser printer is necessary to represent this information. For example, suppose the user asks the following question:

(10) `How do I print out a 10 page file on the laser printer?`

Since the user wants to print a file that KIP knows is 10 pages long on the laser printer, KIP concretes this printing to be an instance of the category PRINTING-LARGE-FILE-ON-LASER-PRINTER. This category has a concern about use of system resources. Thus, printing the file on the laser printer would generate a concern in this particular planning situation. Sending the same file to the lineprinter, however, would not cause a concern since the plan would not be an instance of that category.

10.4.4 Plan Goal Conflict Concerns vs. Effect Goal Conflict Concerns

The previous goal conflict concerns have all been *plan goal conflict concerns*, concerns that a selected plan will conflict with an interest of the user because of a conflicting goal of that plan. These concerns are represented as relations between the plan, the conflicting effect, and the threatened interest. However, there are also concern relations between effects and interests of the user, called *effect goal conflict concerns*. Since sometimes one has concerns about certain effects that are independent of a particular plan, this is an important distinction. For example, deleting a file is an effect that will cause a conflict-of-interest with preserving files of the user, regardless of the plan with which it is associated.

Effect goal conflict concerns also refer to potential conflicts-of-interest among goals of the user. For example, if the user asks:

(11) `I want to delete the file junk, but I want to preserve its contents. What should I do?`

KIP detects a goal conflict between the explicit goal of deleting the file junk, and the explicit goal of preserving the contents of the file junk. The deletion goal is not expressed as the effect of a particular plan. Nonetheless, KIP uses its information about the effect goal conflict concern for the deletion goal. KIP detects the goal conflict because file deletion has a stored conflict-of-interest concern with the goal of file preservation. KIP is able to use all the information it has about this concern outside the context of a particular plan. KIP can use all this information because it considers an effect goal conflict concern in addition to a plan goal conflict concern.

KIP sometimes uses different degrees of concerns for plan goal conflict concerns and effect goal conflict concerns. This technique is useful in situations KIP has information that the conflicting effect is likely. For example, consider again these examples which were discussed earlier in this chapter on page 146:

(2) `How do I print out a file on my terminal?`

(3) `How do I print out an executable file on my terminal?`

In both these examples, the USE-CAT-COMMAND plan might be chosen as a plan for the goal of printing the file on the user's terminal. One potential effect of this plan is that a file with control characters will lock up the user's screen. This effect conflicts with the user's interest in being able to use his terminal effectively. The difference between these examples is the degree of concern regarding this potential goal conflict. In (2), the degree of concern regarding for the screen-locking concern is low. The degree of concern is low because it is very unlikely that the user's file contains control characters. In (3), the degree of concern is high due to the high likelihood that an executable file contains control characters. In both these case, the serious of the threat to the threatened interest is high. The difference in the examples is in the likelihood of the occurrence of the conflicting effect.

In the section on specifity of concern, I described how KIP might create a specific goal/plan relationship that reflects this heightened degree of concern in printing an executable file. However, another way to encode this knowledge is to have different degrees of concern for the plan goal conflict concern and the effect goal conflict concern. According to this technique, KIP stores a low degree of concern for the plan goal conflict concern. This degree is low because in most cases, the conflicting effect is an unlikely occurrence. However, the degree of the effect goal conflict concern is high. The degree of effect goal conflict concern is only influenced by the seriousness of the threat to the user's goal. Since the threat to the user's interest in the usability of the user's terminal is serious, the degree of the effect goal conflict concern is high.

Therefore, if KIP has information that the conflicting effect is likely, it would use the degree of concern information from the potential conflicting effect rather than from the plan itself. For example, in (3) , KIP has knowledge that the conflicting effect is likely. Therefore, KIP infers a high degree of concern due to the high degree of the effect goal conflict concern.

10.5 KIP's Algorithm for Dealing with Goal Conflict Concerns

There are three main parts of KIP's goal conflict concern algorithm:

(1) Concern Retrieval - retrieve the concerns from KIP's planning knowledge base

(2) Concern Evaluation - evaluate the concerns in the particular planning situation

(3) Concern Treatment - decide how the planning process should proceed based on the concern information

First, the three parts of this process are described. I then describe some of the implementation and representation details. In the last part of this section, a short computer trace is presented. In the following chapter, a number of extended examples of both condition concerns and goal conflict concerns will be presented.

In the Figure 10.1, I have expanded on those parts of the KIP's planning algorithm in which goal conflict concerns play an important role.

10.5.1 Concern Retrieval

After KIP detects the goals of the user, it selects a potential plan and creates an instance of that plan. KIP then checks for any violated defaults in the particular planning situation by comparing the values of properties in the planning situation, that have been specified by the user, against the default values for those properties. For each violated default, KIP determines the most specific stored violated default concerns for that violated default. Some violated defaults may generate concerns regarding conflicts due to an effect that is not part of the potential plan. Therefore, the conflicting effects of goal conflict concerns are matched against the effects of the potential plan. KIP discards all concerns whose effects are not effects of the potential plan.

KIP next evaluates whether the user goal is the intended effect of the selected plan. If the goal is not the intended effect of the plan, then both the intended and unintended goal conflict concerns are gathered. If the user goal is the goal for which the plan was intended, then only the intended effect concerns are gathered.

Once intended, unintended and violated default concerns are gathered, it sorts them based on the degree of concern. KIP then decides on a threshold level for concern. This level is based on the planning situation. For example, if the plan is the normal plan for these goals, a high threshold will be chosen. A lower threshold is chosen when the plan has not been used before. The concerns which are below the threshold level are discarded.

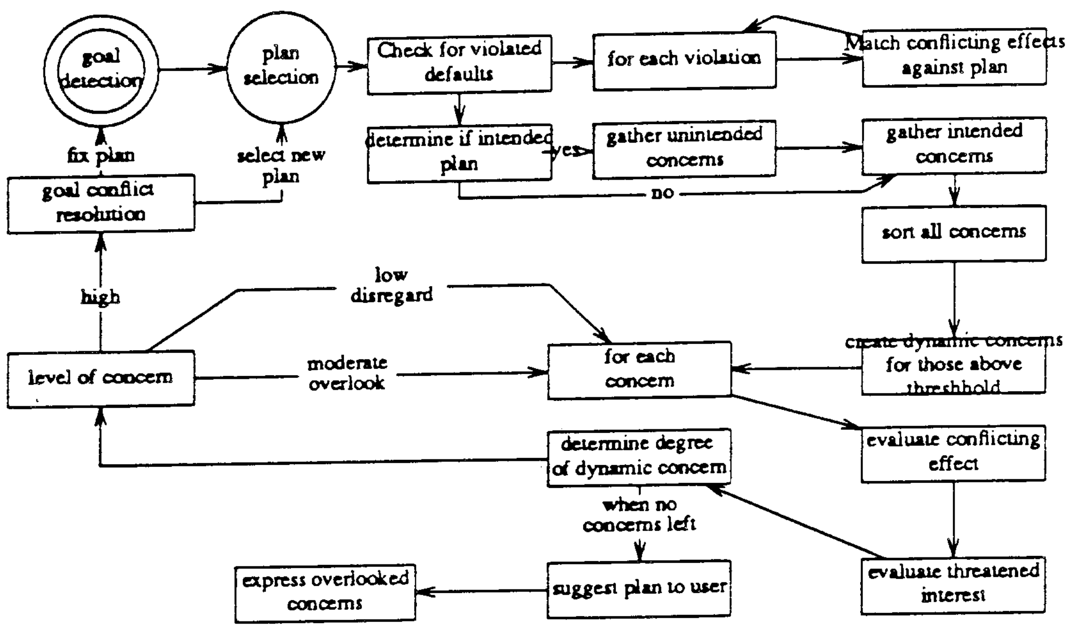

Figure 10.1: KIP's Goal Conflict Concern Algorithm

10.5.2 Concern Evaluation

KIP then creates dynamic concerns for each of the stored concerns. It evaluates the concerns according to the degree of stored concern. KIP first evaluates the conflicting effect by determining if the conditions necessary for the effect are true in KIP's model of the world. Secondly, KIP evaluates the threatened interest to determine if the interest is important in this particular problem situation. Interest evaluation is done by examining the goal that such an interest would generate in the particular planning situation. If the user already has a goal that is mutually exclusive to the interest goal, the interest goal is not generated. If KIP determines that the interest is important, then the interest is inferred as a goal. If not, then the concern is disregarded. In either case the interest evaluation is remembered so that other concerns which are related to this interest are not reevaluated.

During this evaluation, KIP assigns a new degree of concern to the dynamic concern based on the particular planning situation. However, many of the values necessary for this evaluation will not be known and must be provided from uncertain default knowledge. Therefore, the degree of concern of the dynamic concern is calculated by using both the degree of concern of the stored concern and the degree of certainty in the default knowledge. For example, consider a case where KIP evaluates a dynamic concern which has a high degree of concern, and the default knowledge claims that the interest is an unlikely goal. In this case, KIP decides that the degree of concern of the dynamic concern is moderate.

10.5.3 Concern Treatment in the Planning Process

Once KIP has evaluated a concern it can proceed in one of three ways, depending on the degree of that particular concern. If the degree of concern is *low*, KIP can choose to *disregard* the concern. *Disregard* means that the concern is no longer considered at all. KIP can try to revise other parts of the plan, and suggest the plan to the user with no reservations.

If the degree of concern is *high*, KIP can choose to *elevate* the concern to a source of plan failure. In this case, KIP determines that it is very likely that the plan will fail. KIP tries to fix this plan in order to change the value of this condition, or tries to find another plan.

The most complex case is when the degree of concern is *moderate*. In this case, KIP can choose to *disregard* the concern, or *elevate* it to a source of plan failure. KIP can also choose to *overlook* the concern. Once KIP has developed a complete plan for this problem, it is once again faced with the need to deal with the *overlooked* concern. If the plan will work, except for the overlooked concern, KIP can again choose to *disregard* the concern, or *elevate* it to a source of plan failure. At this point, KIP can also choose to suggest an answer to the user. Depending on the degree of this overlooked concern, KIP may choose to express the concern to the user in the answer.

10.5.4 Implementation and Representation

Stored goal conflict concerns are represented in a way very similar to condition concerns as is described in Section 8.6.4. by creating a different GOAL-CONFLICT-CONCERN concept for each concern. Degrees of concerns are presently implemented as numbers from one to ten. Dynamic goal conflict concerns are implemented as instances of these stored concerns.

There are two major differences between the representation/evaluation of condition concerns and goal conflict concerns. Condition concerns have a Desired-Value relation which refers to the value that the Concern-Condition should have. If the Desired-Value does not hold, a goal is instantiated to reflect the condition concern. Goal conflict concerns, however, have an Undesirable-Value relation which refers to the value that the Concern-Condition should not have. Goals are only created to reflect goal conflict concerns when some particular state of the world holds, i.e. the Concern-Condition has the Undesirable-Value. For example, consider the LS-TOO-MUCH-OUTPUT-CONCERN represented in Figure 10.2. If the directory-size is large, then the goal conflict will be considered further. Otherwise the goal conflict will not occur.

The second difference between the representation/evaluation of condition concerns and goal conflict concerns is the reference of goal conflict concerns to interests. While goals are instantiated directly to reflect condition concerns, goal conflict concerns necessitate the evaluation of an interest before a goal is instantiated. Thus, each goal conflict concern also represents a Concern-Interest relation which refers to the interest that will be evaluated when the goal conflict concern is considered.

If the interest is important to the user, a goal is instantiated to reflect the interest. The goal instantiated is defined by the Interest-Derived-Goal relation. In Figure 10.2, the Interest-Derived-Goal is the NO-SCROLL-OFF-SCREEN goal.

10.5.5 Trace of Goal Conflict Concerns

In this section, I present some examples of KIP's use of a goal conflict concern. I first present a long trace of goal conflict concerns. In order to keep the example simple, the only concern is a default concern. However, this same concern is considered at two different points in the planning process, for two different plans with different results.

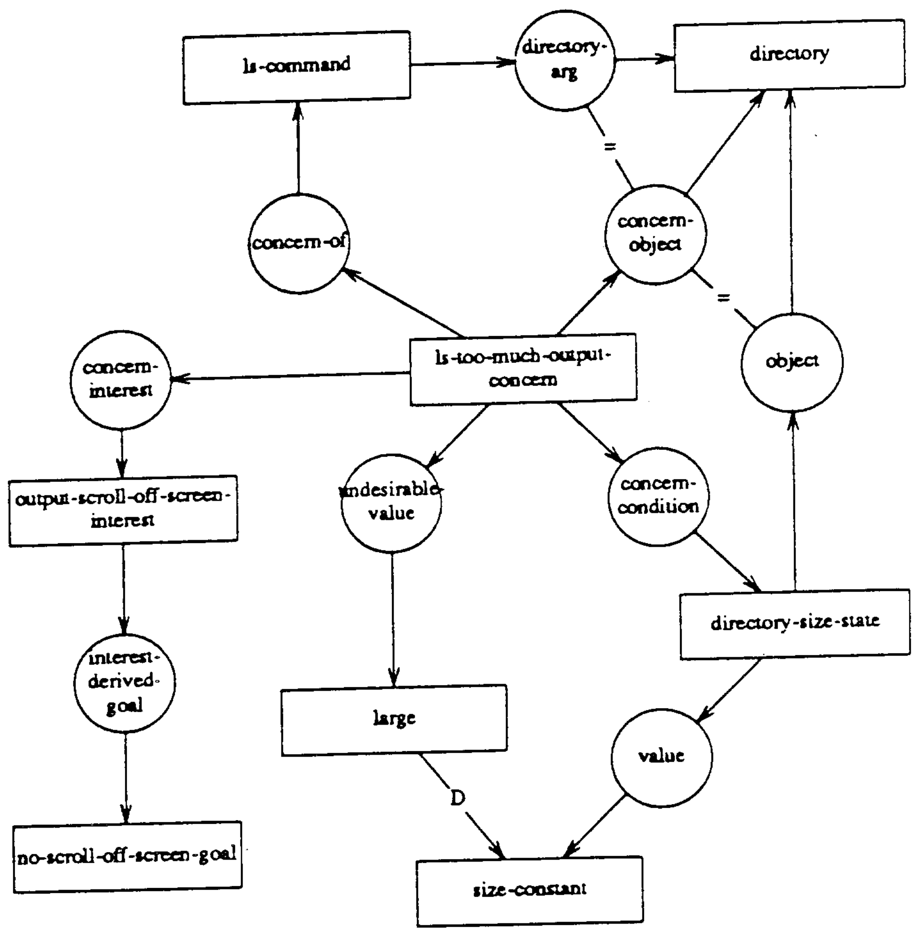

Figure 10.2: Representation of LS-TOO-MUCH-OUTPUT-CONCERN

Figure 10.3: KIP Trace of a Goal Conflict Concern

User: **How I do move the file junk.lisp to the file test1?**

;; KIP has received the following goal as input from the goal analyzer:

```
(rename-file-effect-1
 (Destination-File-File-Name-46
  (file-name-state-38
   (Value-Of-File-Name-50 test1-1)
   (Object-Of-File-Name-42
    (file-2
     (File-Name-41 test1-1)))))
```

;; *The* Destination-File-File-Name-46 *relation is necessary because the*
;; *rename-file-effect manipulates the name*
;; *of the file. After the effect the file referred to by the*
;; *destination-file-file-name is the source file and not the destination file. The*
;; *destination-file-file-name refers to a file-name-state before the effect occurs.*

```
 (Destination-File-Of-Rename-File-Effect-48 file-2)
 (New-Name-37 test1-1)
 (Previous-Name-63 junk.lisp-1)
 (Final-State-Of-Rename-File-Effect-33
  (file-name-state-25
   (Value-Of-File-Name-52 test1-1)
   (Object-Of-File-Name-29 file-1)))
 (Initial-State-Of-Rename-File-Effect-61
  (file-name-state-53
   (Value-Of-File-Name-65 junk.lisp-1)
   (Object-Of-File-Name-57 file-1)))
 (Source-File-Of-Rename-File-Effect-24 file-1))
```

;; the source-file-of-rename-file-effect is the file that is affected in this
;; representation of the rename-file-effect. Its name is changed form junk.lisp to
;; test1.

KIP is trying to determine a plan for the list of goals:
(rename-file-effect-1)

Goal establishment

Selecting a goal from the List Of Goals ((rename-file-effect-1))
selecting the remaining goal
 rename-file-effect-1
Plan determination
Looking for a plan for the Current Goal (rename-file-effect-1)

First looking at stored plans
Selected mv-command as a potential plan
Now specifying the plan for the particular planning situation:
(mv-command-69
 (First-File-Arg-File-Name-98

(file-name-state-90
(Value-Of-File-Name-96 junk.lisp-1)
(Object-Of-File-Name-94 file-1)))
(Second-File-Arg-File-Name-81
(file-name-state-73
(Value-Of-File-Name-79 test1-1)
(Object-Of-File-Name-77 file-2)))

;; These are the file-names before the command is executed. The mv-command is
;; difficult to represent because the argument of the command, i.e. the names of
;; the files, are changed during the course of the plan. Therefore, it is
;; necessary to specify the file-names of the operated files before the command is
;; actually executed

(First-File-Arg-Of-Mv-Command-89 file-1)
(Second-File-Arg-Of-Mv-Command-72 file-2)
(Format-Of-Unix-Two-File-Command-87
(unix-two-file-command-format-86
(Command-Arg-106 mv-string-103)
(Format-First-File-Arg-100 junk.lisp-1)
(Format-Second-File-Arg-108 test1-1)))

;; The user could execute this plan by typing mv junk.lisp test1

(Intended-Effect-Of-Mv-Command-70 rename-file-effect-1))

;; The representation of rename-file-effect-1 *is shown above*

Plan failure detection

;; KIP is now looking at the concerns of this plan.

No violated default concerns

Evaluating the goal conflict concern: delete-destination-file-concern-119

;; This concern is a general concern about a class of UNIX commands that delete
;; the destination file.

This Concern (delete-destination-file-concern-119) potentially conflicts
with the users Concern Interest (file-preservation-interest-due-to-name-overwrite-140)

;; This concern causes KIP to consider this threatened interest. The
;; file-preservation-interest is the interest in preserving the user's files. The
;; current interest is a more specific category of that interest which refers to
;; interests that were threatened due to name overwrite.

(file-preservation-interest-due-to-name-overwrite-140
(Interest-Object-197 file-2)
(Concern-Concern-Of-143 mv-command-69))

;; However, this interest is only applicable if certains conditions hold

This interest is applicable depending on the Concern Condition (file-exist-state-125) of file-2
The current value of the condition is truth-value-127
The undesirable value is true
Current Value (truth-value-127) is less specific than the Undesirable Value (true)
Try to make the current value more specific using defaults
Default-value is anything.

The Current Value (truth-value-127) is more specific than the Default Value (anything), Therefore it is somewhat likely that the concern is important

Kip is now determining whether a goal should be instantiated to reflect the user Interest (file-preservation-interest-due-to-name-overwrite-140)

Since the degree of concern is medium, a goal is instantiated which reflects the threat to this interest in this planning situation

;; Since KIP knows the nature of the threat to the user's goal, KIP also instantiates
;; a goal which reflects this threat. In this case, KIP has stored that in order to
;; address this threat to the user's interest, a goal is instantiated of renaming the
;; threatened file to a file with a .old extension.

```
Creating the goal:
(concern-generated-goal-198
 ;; This is actually a rename-file-effect goal but the fact that is generated by
 ;; a concern is also stored
 (Destination-File-File-Name-185
  (file-name-state-177
   (Value-Of-File-Name-189
    (old-file-name-192
     (Old-Name-Part-163 test1-1)))
   ;; An old-file-name is the the old-name-part followed by .old. In this case, the
   ;; name is test1.old
   (Object-Of-File-Name-181
    (file-178
     (File-Name-180 old-file-name-192)))))
 (Destination-File-Of-Rename-File-Effect-187 file-178)
 (New-Name-Of-Rename-Old-File-Effect-159 old-file-name-192)
 (Previous-Name-Of-Rename-Old-File-Effect-161 test1-1)
 (Final-State-Of-Rename-File-Effect-155
  (file-name-state-147
   (Value-Of-File-Name-191 old-file-name-192)
   (Object-Of-File-Name-151
    (file-2
     (File-Exist-128 truth-value-127)
     (File-Name-167 test1-1)
     (File-Name-150 old-file-name-192)))))
 (Initial-State-Of-Rename-File-Effect-172
  (file-name-state-164
   (Value-Of-File-Name-176 test1-1)
   (Object-Of-File-Name-168 file-2)))
 (Source-File-Of-Rename-File-Effect-174 file-2)
 (State-Change-Interval-Of-Rename-File-Effect-157 state-change-time-156))
```

Asserting the fact that the final-state of the goal (file-name-state-147) starts before the start of plan interval (mv-command-69)
So that the condition holds before the plan is executed

Goal establishment
Selecting a goal from the List Of Goals ((concern-generated-goal-198))

;; KIP has added the concern-generated goal to its list of goals and is now
;; planning for that goal

```
selecting the remaining goal
 concern-generated-goal-198
Plan Determination
Looking for a plan for the Current Goal (concern-generated-goal-198)
First looking at stored plans
Selected mv-command as a potential plan
Now specifying the plan for the particular planning situation:
(mv-command-199
 (First-File-Arg-File-Name-228
  (file-name-state-220
   (Value-Of-File-Name-226 test1-1)
   (Object-Of-File-Name-224 file-2)))
 (Second-File-Arg-File-Name-211
  (file-name-state-203
   (Value-Of-File-Name-210
    (old-file-name-192
     (Old-Name-Part-163 test1-1)))
   (Value-Of-File-Name-209 old-file-name-192)
   (Object-Of-File-Name-207 file-178)))
 (First-File-Arg-Of-Mv-Command-219 file-2)
 (Second-File-Arg-Of-Mv-Command-202 file-178)
 (Format-Of-Unix-Two-File-Command-217
  (unix-two-file-command-format-216
   (Command-Arg-236 mv-string-233)
   (Format-First-File-Arg-230 test1-1)
   (Format-Second-File-Arg-238 old-file-name-192)))
```

;; The user could execute this plan by typing mv test1 test1.old

```
 (Intended-Effect-Of-Mv-Command-200 concern-generated-goal-198))
Asserting that mv-command-199 comes before mv-command-69
```

;; This type of threat to the user's plan must be addressed before the plan is
;; executed.

```
Plan failure detection
Evaluating the goal conflict concern: delete-destination-file-concern-251
```

;; KIP has detected the same concern regarding this new plan

```
This Concern (delete-destination-file-concern-251) potentially conflicts
with the users Concern Interest (file-preservation-interest-due-to-name-overwrite-272)
This interest is applicable depending on the Concern Condition (file-exist-state-257) of file-178
The current value of the condition is truth-value-259
The undesirable value is true
 Current Value (truth-value-259) is less specific than the Undesirable Value (true)
Try to make the current value more specific using defaults
Default-value is anything.
The Current Value (truth-value-259) is more specific than the Default Value (anything),
Therefore it is somewhat likely that the concern is important
```

;; KIP has determined that the file test1.old might already exist and would thus be deleted

Kip is now determining whether a goal should be instantiated to
reflect the user Interest (file-preservation-interest-due-to-name-overwrite-272)
Since the degree of concern is medium a goal is instantiated which reflects the threat to this interest
In this planning situation
Kip has determined that the Interest (old-file-preservation-interest-due-to-name-overwrite-332)
is not important in this particular planning situation

;; KIP has realized that this is actually another type of file-preservation-interest,
;; the old-file-preservation-interest. This category refers to the interest in
;; preserving old-files. KIP knows that the user's level of interest in preserving
;; old-files is relatively low. User's tend to keep one .old version around and no
;; more.

The following goal will not be instantiated:
(rename-old-file-effect-278)

To move the file named junk.lisp to the file named test1, use mv junk.lisp test1.
However, the file test1 will be deleted if the file test1 exists.
First move the file named test1 to the file named test1.old, by typing mv test1 test1.old.

In the following trace, KIP determines a 4-step plan from concerns presented in the previous trace.

Figure 10.4: KIP Trace of File Swapping

User: How do I move the file foo to the file named bar,
and move the file named bar to the file named foo?

To move the file named foo to the file named foo.old, use mv foo foo.old.
To move the file named bar to the file named bar.old, use mv bar bar.old.
To move the file named bar.old to the file named foo, use mv bar.old foo.
To move the file named foo.old to the file named bar, use mv foo.old bar.

;; The user has asked how to swap two files. However, KIP has no knowledge about
;; swapping. KIP arrives at this solution by using the same concerns presented in
;; the previous example. KIP first decides to move foo to bar, but realizes that bar
;; will be deleted. Therefore, KIP decides to move bar to bar.old. KIP plans
;; similarly for the move from bar to foo. KIP then determines what the order of the
;; plans should be and returns a plan to the user. KIP's plan is suboptimal, it could
;; have determined a plan with three steps rather than four. However, KIP has not
;; been given strategies for reducing the number of plan steps.

In the following trace, an unintended effect concern trace is presented.

Figure 10.5: KIP Trace for Freeing up Disk Space

User: How can I free up disk space?

To free up disk space, use rm.
However, any file that is removed cannot be recovered.

;; KIP has not been given a plan for the goal of freeing up disk space. However,
;; freeing up disk space is one of the effects of the rm command. KIP selects the
;; rm-command as a plan for the goal of the user. However, KIP expresses concern
;; that the user cannot recover his files once they have been deleted. This is an
;; example of an unintended effect concern. Unintended effect concerns are only
;; considered when the user's goal is not the intended effect of the selected plan.

10.6 Conclusion

This chapter has focused on goal conflict concerns. KIP's use of goal conflict concerns allows KIP to detect complex goal conflicts between an effect of a potential plan and a goal or interest of the user. Eight different dimensions of concern have been described that allow KIP to detect goal conflicts in unique planning situations.

Chapter 11

Conclusion

11.1 Importance of Concerns

The earlier planners described in Chapter 1, planned without representing the concerns of a plan. In the course of this thesis, I have demonstrated the use of concerns in planning for goals in many different planning situations. This section addresses the relative advantages of concern representation and processing. In particular, the gains in efficiency resulting from concerns are evaluated in light of the processing demands such concern representation entails.

11.1.1 Efficiency

Concerns allow a planner to plan efficiently in planning situations where the planner has much knowledge about potential plan failures. As the planner gains more knowledge regarding potential plan failures in a particular domain, planning becomes computationally intractable using other methods. Thus, concerns allow the planner to plan in domains where planning might become impossible for traditional methods.

Knowledge-deficient planners actually implemented de facto concerns. Many of the conditions represented in knowledge-deficient planners are those that are most likely to fail in a particular planning situation. Since knowledge-deficient planners do not use the concern vocabulary, they fail to distinguish between conditions more likely to fail and conditions less likely to fail.

11.1.2 Planning with Uncertainty

A commonsense planner must be able to detect plan failures even when it is *uncertain* of values in the planning situation. Most previous planning research assumed that

the values for all the conditions is known. However, in UC, when a user describes a planning problem later passed to KIP, the values for many conditions are usually omitted. It would be undesirable to prompt the user for this information, particularly for those values which are not important for the specific planning situation. Instead, an algorithm for plan failure detection should be able to detect plan failures by relying on default situation knowledge. The use of default situation knowledge may entail further processing for all the plan failures considered by a planner. Therefore, it becomes even more important to limit the conditions, effects, goals, and interests a planner considers. Since much of the knowledge about a particular situation may be unknown, each individual comparison for conflict might entail much effort.

Concerns allow a planner to plan in uncertain planning situations. A planner only considers violated default concerns when specified parameters violate defaults. Thus, default/violated-default concerns allow KIP to deal with a large body of default knowledge. Concerns circumvent the need to refer to default knowledge regarding the value of those conditions about which KIP is unconcerned.

11.1.3 Knowledge Intensive

The third reason for using concerns in a knowledge intensive planner is based on the principles which underlie knowledge intensive planning. Knowledge intensive planning implies the representation and efficient utilization of as much knowledge as possible. Concerns are a type of knowledge, like any other knowledge that a planner might have. Human experts know about plans, goals, interests, etc. Similarly, they also know about concerns. Therefore, concerns should be modeled by a knowledge intensive planner. The fact that a knowledge intensive planner has a large knowledge base is problematic due to the potential problems of focusing on relevant knowledge. While other types of knowledge tend to diffuse the planner's attention by introducing more possibilities in a planner's search space, concerns focus the planner's attention. Therefore, concerns should not only be modeled, they should be utilized.

11.2 Future Research

Hammond's (CHEF) [13] method of indexing plans on the failures that such plans avoid might be a useful addition to the theory of concerns. In KIP, goals are instantiated that reflect concerns, and KIP selects plans for these goals. Research should further address the relationship between concerns about particular plan failures and plans that avoid such failures. Also, KIP's concerns are currently defined by explicitly representing concerns in a long term knowledge base based on a human consultant's UNIX experience. Hammond's method of learning plan failures might be useful in investigating the learning of concerns

through the examination of plan failures in previous cases.

KIP - the Theory and KIP - the Program might be modified so as to reflect Wilensky's [35] theory of metaplanning. This modification would provide a specification of planning strategies. In particular, KIP's concern algorithm might benefit from meta-level knowledge regarding the detection of plan failures. Such knowledge might be particularly useful where specific concern knowledge regarding a planning situation is not available.

11.3 Final Comments - Building Knowledge-Based Systems

KIP - the Program is actually a relatively simple program. Eighty percent of my work in implementing KIP was in the development of KIP's knowledge base. This is particularly appropriate for a knowledge based program such as KIP, but has been the cause of much frustration. Unlike a computer program, where one can use a compiler as a first pass debugger, there are no readily available tools for debugging a knowledge base.

There are a number of simple strategies that I have found useful in developing such a knowledge base. The simplest strategy is to maintain a list of test cases that have worked in the past. As new knowledge is added, these examples should be retried. Unfortunately, one can also depend too heavily on such strategy. Occasionally, new information added to the knowledge base showed that one or more test cases were not represented correctly due to incorrect assumptions.

Another strategy for proper knowledge base construction is the checking of constraints in the knowledge base when the knowledge base is loaded. In addition to enforcement of constraint checking on the part of the knowledge base interpreter, relations must be carefully constrained by the knowledge base designer.

One of the best strategies I found for discovering knowledge base inconsistencies was the application of the knowledge base for other tasks. Many errors were detected by using the same knowledge base for planning as for parsing questions to UC.

The real test for any knowledge based program is applying the program to a great deal of knowledge. However, as knowledge bases grow larger new strategies for knowledge base construction are necessary. Tools should be developed that help an expert add information to a knowledge base such as KODIAK. These tools should immediately point out inconsistencies in the knowledge base. For example, one of the most difficult tasks in constructing KIP's knowledge base was the construction of classification hierarchies. Tools are needed to display the network in such a way as to aid the knowledge base designer in the construction of these hierarchies.

Bibliography

[1] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[2] J. F. Allen and P. J. Hayes. A common-sense theory of time. In *Proceedings of the 9th International Joint Conference on Artificial Inteligence*, Los Angeles, CA, August 1985.

[3] J. F. Allen and P. J. Hayes. Short time periods. In *Proceedings of the 10th International Joint Conference on Artificial Inteligence*, Milan, Italy, August 1987.

[4] R. Alterman. An adaptive planner. In *Proceedings of the 5th National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986.

[5] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[6] David Ngi Chin. *Intelligent Agents as a Basis for Natural Language Interfaces*. Report UCB/CSD 87/396, University of California, Berkeley Electronic Research Laboratory, 1987.

[7] David Ngi Chin. KNOME: modeling what the user knows in UC. In Alfred Kobsa and Wolfgang Wahlster, editors, *User Models in Dialog Systems*, Springer-Verlag, Berlin, 1988.

[8] C. A. Cox. *ALANA: Augmentable LANguage Analyzer*. Report UCB/CSD 86/283, University of California, Berkeley Electronic Research Laboratory, 1986.

[9] G. F. Dejong and R. Mooney. Explanation-based learning: an alternative view. *Machine Learning*, 1:145–176, 1986.

[10] G. Ernst and A. Newell. *GPS: A Case Study in Generality and Problem Solving*. Academic Press, New York, 1969.

[11] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalize robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[12] R. E. Fikes and N. J. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[13] K. Hammond. *Case-based Planning: An Integrated Theory of Planning, Learning, and Memory*. Department of Computer Science 488, Yale University, 1986.

[14] J. Hendler. *Integrating Marker-Passing and Problem-Solving*. Technical Report CS-85-08, Brown University, 1985.

[15] M. Luria. Commonsense planning in a consultant system. In *Proceedings of 9th Conference of the IEEE on Systems, Man, and Cybernetics*, Tuscon, AZ, November 1985.

[16] M. Luria. Concerns: a means of identifying potential plan failures. In *Proceedings of the Third IEEE Conference on Artificial Intelligence Applications*, Orlando, Florida, February 1987.

[17] M. Luria. Concerns: how to detect plan failures. In *Proceedings of the Third Annual Conference on Theoretical Issues in Conceptual Information Processing*, Philadelphia, PA, August 1986.

[18] M. Luria. Dividing up the question answering process. In *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, PA, August 1982.

[19] M. Luria. Expressing concern. In *Twenty Fifth Annual Meeting of the Association for Computational Linguistics*, Stanford, CA, July 1987.

[20] M. Luria. Goal conflict concerns. In *Proceedings of the Tenth International Joint Conference on Artificial Inteligence*, Milan, Italy, August 1987.

[21] M. Luria. Question answering: two separate processes. In *Proceedings of the 4th National Conference of the Cognitive Science Society*, Ann Arbor, MI, August 1982.

[22] J. Martin. Knowledge acquisition through natural language dialogue. In *Proceedings of the 2nd Conference on Artificial Intelligence Applications*, Miami, Florida, 1985.

[23] J. Martin. Understanding new metaphors. In *Proceedings of the Tenth International Joint Conference on Artificial Inteligence*, Milan, Italy, August 1987.

[24] J. Martin. Views from a kill. In *Proceedings of the Seventh Annual Meeting of the Cognitive Science Society*, Amherst, MA, 1986.

[25] J. Mayfield. When to keep thinking. In *Proceedings of the Third Annual Conference on Theoretical Issues in Conceptual Information Processing*, Philadelphia, PA, 1986.

[26] J. McCarthy. *Situations, Actions, and Causal Laws*. Stanford Artificial Intelligence Memo 2, Stanford University, Stanford, CA, 1963.

[27] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[28] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1:47–80, 1986.

[29] A. Newell and H. A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[30] Peter Norvig. *A Unified Theory of Inference for Text Understanding*. Report UCB/CSD 87/339, University of California, Berkeley Electronic Research Laboratory, 1987.

[31] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[32] E. Sacerdoti. *A Structure for Plans and Behavior*. Elsevier North-Holland, New York, 1977.

[33] R. Schank and R. P. Abelson. *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum, Hillsdale, NJ, 1977.

[34] G. J. Sussman. *A Computer Model of Skill Acquisition*. American Elsevier, New York, 1975.

[35] R. Wilensky. *Planning and Understanding: A Computational Approach to Human Reasoning*. Addison-Wesley, Reading, MA, 1983.

[36] R. Wilensky. *Some Problems and Proposals for Knowledge Representation*. Memorandum UCB/CSD 87/351, University of California, Berkeley Electronic Research Laboratory, 1987.

[37] R. Wilensky. *Understanding Goal-Based Stories*. Research Report 140, Yale University, 1978.

[38] R. Wilensky, J. Mayfield, D. Chin, C. Cox, M. Luria, M. Martin, and D. Wu. *UC - A Progress Report*. Memorandum UCB/CSD 87/303, University of California, Berkeley Electronic Research Laboratory, 1987.